P.1877/87

6 1987

informatyka

Problematyka systemów operacyjnych

Systemy ekspertowe na IBM PC

WYDAWNICTWO NOT SIGMA

Cena 150 zł

informatyka

6 1987

Nr 6
Miesięcznik Rok XXI
Czerwiec 1987

Organ Komitetu
Naukowo-Technicznego NOT
ds. Informatyki

KOLEGIUM REDAKCYJNE:

Dr inż. Wacław ISZKOWSKI, mgr Teresa JABŁOŃSKA (sekretarz redakcji), Władysław KLEPACZ (redaktor naczelny), dr inż. Marek MACHURA, Maria PAWLAK (sekretarz redakcji), mgr inż. Jan RYŻKO, dr inż. Wiktor RZECZKOWSKI, mgr Hanna WŁODARSKA, dr inż. Janusz ZALEWSKI (zastępca red. naczelnego)

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab. Juliusz Lech KULIKOWSKI

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 01-517 Warszawa, ul. Mickiewicza 18 m. 17, tel. 39-14-34

Zakł. Graf. „Tamka". Zam. 0278-1300/87. Obj. 4,0 ark. druk. Nakł. 8050 egz. K-82.

ISSN 0542-9951. INDEKS 36124

Cena egzemplarza 150 zł
Prenumerata roczna 1800 zł

WYDAWNICTWO
NAUKOWO-TECHNICZNE
SIGMA
CZASOPISM I KSIĄŻEK TECHNICZNYCH
00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE:

Strona

Problematyka systemów operacyjnych na przykładzie systemu Unix (1)
Jan Madey 1

Logika w systemach ekspertowych (2)
Robert Kowalski 5

Grafika żółwia w języku Forth (2)
Zbigniew Szkaradnik 8

Synchronizacja transakcji w systemach rozproszonych baz danych (2)
Tadeusz Morzy 11

Adaptacja drukarki wierszowej DW 204-2 do pracy z minikomputerem SM-4
Czesław Ratka 14

Concurrent DOS (2)
Janusz Zalewski 17

Kompilator Ady dla mikrokomputerów
Opr. Mariusz Kuc 20

ZE ŚWIATA 25

Systemy ekspertowe na mikrokomputery IBM PC

Entity — Relationship Approach '86

Kto jest kim w IFIP. Heinz Zemanek

TERMINOLOGIA 29

Terminologia polskiego Logo (2)

W NAJBLIŻSZYCH NUMERACH:

- Mary Shaw o kolejnych wyzwaniach dla inżynierii oprogramowania
- Ryszard Tadeusiewicz o głosowym wprowadzaniu informacji do komputera
- Jan Bielecki o języku Turbo Pascal
- Anna Syfert, Andrzej Rażnowiecki i Janusz Zalewski o strukturze systemu operacyjnego PC-DOS (1)
- Mariusz Kuc o kompilatorach Ady na komputery IBM PC
- Janusz Zalewski o obszarze komunikacyjnym systemu BIOS

JAN MADEY

Uniwersytet Warszawski

POLSKIE TOWARZYSTWO INFORMATYCZNE

Pierwsza wersja artykułu została przedstawiona podczas Szkoły Jesiennej PTI „Współczesne kierunki rozwoju informatyki" — Mrągowo, listopad 1986 r.

P. 1877/87

Problematyka systemów operacyjnych na przykładzie systemu Unix (I)

Niezbędnym elementem oprogramowania podstawowego każdego komputera (mikrokomputera) jest **system operacyjny.** Jego zadaniem jest tworzenie środowiska operacyjnego, w którym użytkownik może wygodnie i bezpiecznie opracowywać, uruchamiać lub eksploatować programy. System operacyjny ukrywa przed użytkownikiem pewne cechy i funkcje sprzętowe, a dostarcza mu w zamian narzędzi wygodniejszych do rozwiązywania problemów. W szczególności system operacyjny umożliwia:

- przechowywanie informacji przez dłuższy czas w pamięciach zewnętrznych,
- wspólne wykorzystywanie niektórych urządzeń komputerowych przez grupy osób,
- jednoczesne wykonywanie różnych czynności przez system komputerowy,
- korzystanie z różnorodnych pakietów programistycznych i języków programowania.

W zależności od typu komputera, trybu jego wykorzystywania oraz od rodzaju urządzeń wchodzących w skład instalacji, systemy operacyjne istotnie różnią się od siebie stopniem złożoności i zakresem oferowanych usług. Znane są systemy operacyjne, które były konstruowane przez kilka lat w wieloosobowych zespołach specjalistów i mają bardzo złą reputację (np. OS/360 dla komputerów typu IBM 360/370). Znane są również systemy, które powstały bardzo szybko, w małych zespołach, a mimo to przebiły się i wzbudziły powszechne zainteresowanie. Najważniejszym reprezentantem tej grupy jest system **Unix.** Chociaż Unix ma już ponad piętnastoletnią historię, jego popularność ciągle rośnie, a niektóre z przyjętych w nim rozwiązań są naśladowane w nowszych systemach.

System Unix jest podobnym zjawiskiem wśród systemów operacyjnych, jak język Pascal wśród języków programowania. W obu wypadkach był do produkt autorstwa małej grupy osób, tworzony bez reklamy i specjalnego poparcia. W obu wypadkach hasłem naczelnym była **prostota,** a twórcy świetnie wyczuli przy tym potrzeby odbiorcy. W efekcie ich dzieła odniosły prawdziwy sukces, przebiły się bez instytucjonalnych sponsorów. Co ciekawsze, zarówno Pascal, jak i Unix, stają się obecnie standardami dla mikrokomputerów, mimo że nie były projektowane z myślą o tym typie sprzętu.

Pierwszą wersję systemu operacyjnego Unix opracowano na przełomie lat sześćdziesiątych i siedemdziesiątych w Bell Laboratories, stan New Jersey, USA, na wewnętrzny użytek tej firmy. Unix, napisany początkowo w języku asemblera na komputer DEC PDP-7, tworzył środowisko ułatwiające programowanie. Głównymi odbiorcami i krytykami tej wersji systemu byli sami jego twórcy. Ze względu na potrzebę przeniesienia systemu na inny typ komputera, został on ponownie zaprogramowany w języku wysokiego poziomu (język ten, o nazwie **C**, został opracowany do tego właśnie celu) i zainstalowany na PDP-11. Przy tej okazji wprowadzono do systemu pewne rozszerzenia i modyfikacje. W 1973 roku Unix został udostępniony ośrodkom akademickim, co bardzo szybko przyniosło mu dużą popularność. Z czasem różne instytucje zajęły się rozwijaniem i dystrybucją systemu Unix, a w szczególności przenoszeniem Unixa na inne komputery. Powstało przy tym wiele różnych wersji i mutacji systemu, często dość istotnie różniących się między sobą. Do najbardziej zaawansowanych należą implementacje na komputery VAX opracowane przez oddział Uniwersytetu Kalifornijskiego w Berkeley. Jedną z ostatnich wersji jest **Unix 4.2BSD** (ang. Berkeley Software Distribution).

Interesujący jest również zwrot ku mikrokomputerom. Już obecnie istnieją instalacje Unixa na systemy 16-bitowe z procesorami firm Motorola, Intel i Zilog. Powstają również mutacje Unixa, noszące odmienne nazwy. Na szczególną uwagę zasługują dwie z nich: **Xenix** oraz **Tunis.** Pierwszy z tych systemów jest produktem firmy Microsoft i został wprowadzony na rynek w 1980 roku jako komercyjna wersja Unixa dla mikroprocesorów Intel 8086/88. Zmodyfikowano przy tym nieznacznie strukturę Unixa oraz wprowadzono pewne ulepszenia (Xenix bazował na wersji **Unix V7** z 1979 roku). System Tunis powstał natomiast jako produkt uboczny pewnego projektu na Uniwersytecie w Toronto. Celem tego projektu było wykazanie, że jeżeli dysponuje się właściwymi narzędziami (odpowiednim językiem i metodyką programowania), to zadania programowania systemowego znacznie się upraszczają. W Toronto opracowano specjalny język o nazwie **Concurrent Euclid** i w tym języku zrealizowano — głównie siłami studentów — nową wersję Unixa na komputer VAX, w której została od nowa zaprojektowana najistotniejsza część systemu (tzw. jądro). Tunis jest obecnie przenoszony także na mikrokomputery 16-bitowe. Również inne mikrokomputerowe systemy operacyjne (np. MS-DOS) mają wiele cech przejętych z Unixa. Wszystko więc wskazuje na to, że Unix staje się standardowym systemem operacyjnym dla komputerów 16- i 32-bitowych, a dominują-

Doc. dr hab. **JAN MADEY** w 1964 r. ukończył studia matematyczne na Uniwersytecie Warszawskim. Tamże w 1972 r. uzyskał doktorat, a w 1979 — habilitację. Jest zatrudniony od 1964 r. jako nauczyciel akademicki na Uniwersytecie Warszawskim. Zajmuje się językami programowania, systemami operacyjnymi i dydaktyką informatyki. Jest autorem i współautorem podręczników akademickich (Algol 60 i Pascal) oraz różnych opracowań z dziedziny informatyki. Odbył długotrwałe staże naukowe w Wielkiej Brytanii i RFN. Prowadził także wykłady w USA: na Uniwersytecie Kalifornijskim (oddział w Santa Cruz) oraz w Wang Institute of Graduate Studies (Massachusetts).

Obecnie pełni funkcje: dyrektora Instytutu Informatyki UW, przewodniczącego Zespołu Doradczego ds. Informatyki Ministra Oświaty i Wychowania, członka Zespołu Dydaktyczno-Naukowego Nauk Ścisłych Ministerstwa Nauki i Szkolnictwa Wyższego, członka Rady Naukowej Ogólnopolskiej Fundacji Edukacji Komputerowej, członka Zespołu Doradczego ds. dzieci szczególnie uzdolnionych Krajowej Fundacji na Rzecz Dzieci, członka Komitetu Redakcyjnego serii: Biblioteka Inżynierii Oprogramowania, w Wydawnictwach Naukowo-Technicznych.

cą w tym kontekście wersją jest obecnie tzw. **Unix System V, edycja 2**, z 1984 roku. Dotyczy to także sieci — powstają wersje „rozproszone" Unixa. Należy przypuszczać, że dopiero systemy komputerowe o istotnie nowej architekturze mogą spowodować zmierzch Unixa.

W niniejszym artykule przedstawiono krótko wybrane aspekty problematyki systemów operacyjnych i zilustrowano je na przykładzie systemu Unix. Ze względu na złożoność zagadnienia, prezentacja ta jest bardzo zwięzła i daleka od kompletności.

STRUKTURA SYSTEMU OPERACYJNEGO

Charakterystyczną cechą nowoczesnych systemów operacyjnych jest struktura warstwowa.

Podział systemu na warstwy

Każdy system operacyjny ma wyodrębnioną część, która realizuje jego najbardziej podstawowe funkcje. Część tę nazywa się zwykle **jądrem** systemu albo **warstwą wewnętrzną.** Pozostała część systemu tworzy **warstwę zewnętrzną.** Podział taki nie jest jednoznacznie ustalony dla wszystkich systemów. Wczesne systemy operacyjne miały niekiedy warstwę zewnętrzną pustą, tzn. system był niepodzielny, tworzył jedną całość. Niekiedy autorzy systemu nie prezentowali jego struktury w sposób klarowny lub też stosowali odmienną terminologię. Niemniej jednak wydaje się, że poza patologicznymi przypadkami można zawsze rozróżnić te dwie warstwy, a dla celów dydaktycznych wyodrębnienie ich ma duże znaczenie. W jądrze systemu implementuje się te wszystkie funkcje, które z racji swojej wagi i charakteru muszą być traktowane w uprzywilejowany sposób. W szczególności więc jądro jest odpowiedzialne za:

- obsługę przerwań,
- przydział procesora,
- operacje wejścia-wyjścia.

Ponadto w jądrze są implementowane mechanizmy, dzięki którym system komputerowy wraz z jądrem tworzy bardziej atrakcyjną dla użytkownika maszynę, zwaną często **maszyną wirtualną** lub **maszyną rozszerzoną.** Na przykład, warstwa zewnętrzna może składać się z **procesów współbieżnych** — wówczas jądro systemu musi dostarczyć narzędzi m.in. do tworzenia i usuwania procesów oraz do ich synchronizowania.

Jakie funkcje realizuje warstwa zewnętrzna? Trudno na to pytanie odpowiedzieć jednoznacznie. Zwykle warstwa zewnętrzna obejmuje procesy użytkowników i te moduły systemu operacyjnego, które mogą współzawodniczyć z użytkownikami o zasoby w podobny sposób, jak użytkownicy współzawodniczą między sobą. Inaczej mówiąc, w warstwie zewnętrznej umieszcza się te funkcje systemu, przed których wykonaniem można „trochę poczekać" na procesor, pamięć operacyjną lub inne zasoby. Przykładem jest tu moduł szeregujący zadania (ang. job scheduler) lub moduły odpowiedzialne za wczytywanie zadań i drukowanie wyników (ang. spooler). Warstwa zewnętrzna ma niekiedy postać rodziny procesów współbieżnych, zorganizowanych w **strukturę hierarchiczną.** Relacją określającą tę strukturę jest zwykle zależność zleceniodawca-zleceniobiorca.

Struktura systemu Unix

Unix jest wielodostępnym systemem operacyjnym, wyposażonym ponadto w różne narzędzia wspomagające projektowanie, tworzenie i uruchamianie programów. Narzędzia takie nie stanowią części systemu operacyjnego przy tradycyjnej definicji tego pojęcia. Dlatego też termin „Unix" jest utożsamiony powszechnie nie tyle z samym systemem operacyjnym, co z całym bogatym środowiskiem programistycznym. Wśród ułatwień programistycznych są znane edytory (np. **ed, vi, emacs**), programy wspomagające redagowanie tekstów (np. **troff, scribe**), kompilatory popularnych języków programowania (np. **Pascal, C, Fortran**), programy porównujące teksty (**cmp, diff**), wyszukujące określone sekwencje znaków (**grep, awk**), sortujące (**sort**), obsługujące pocztę elektroniczną (**mail**) itd.

Podział na warstwy jest w Unixie bardzo klarowny. Jądro systemu, poza czynnościami wspomnianymi w poprzednim punkcie (obsługa przerwań, przydział procesora, wejście-wyjście), realizuje bardzo ważne w Unixie pojęcie **procesu.** Ponadto jądro prowadzi gospodarkę pamięcią operacyjną i pomocniczą, a w szczególności realizuje interesujący **system plików.** Każdej akcji wykonywanej poza jądrem odpowiada pewien proces. Warstwa zewnętrzna Unixa jest więc zestawem procesów, które odpowiadają wykonywaniu zarówno wspomnianych uprzednio programów systemowych, jak i programów użytkownika. Wśród procesów warstwy zewnętrznej specjalną rolę odgrywa tzw. **powłoka** (ang. shell). Jest to proces odpowiedzialny za konwersację z użytkownikiem, tzn. interpretujący komendy wprowadzane z klawiatury terminala. Powłoka jest przy tym wymienialna — każdy użytkownik Unixa może napisać swój własny interpretator komend i wstawić go w miejsce standardowego. Obecnie w powszechnym użyciu są dwa interpretatory: tzw. **Bourne shell** (od nazwiska jego twórcy) i **C shell.**

WSPÓŁBIEŻNOŚĆ W SYSTEMACH OPERACYJNYCH

Różne składowe instalacji komputerowej mogą w pewnych okresach pracować równolegle. Podobnie różne programy użytkowników, a także niektóre programy systemowe mogą być jednocześnie wykonywane, o ile pozwalają na to warunki sprzętowe i organizacyjne. Co więcej, w ramach pojedynczego programu dają się często wyodrębnić fragmenty, które można także wykonywać równolegle. Spostrzeżenia te doprowadziły do powstania i rozwoju bardzo ważnej dyscypliny — **programowania współbieżnego.** Zajmuje się ona całokształtem problemów związanych z notacją i technikami służącymi do wyrażania **potencjalnej równoległości.** W szczególności chodzi o rozwiązywanie pojawiających się w tych sytuacjach problemów **synchronizacyjnych i komunikacyjnych.**

Procesy współbieżne

Ścisłe określenie pojęcia „proces" jest trudne. Trzeba w tym celu posłużyć się precyzyjnym aparatem matematycznym. Dlatego też w podręcznikach poświęconych systemom operacyjnym oraz w wielu pracach na ten temat stosuje się zwykle intuicyjną, nieformalną definicję.

Proces sekwencyjny jest to realizacja programu sekwencyjnego lub też wstrzymana realizacja w oczekiwaniu na zdarzenie, które umożliwi jej kontynuację. Proces odpowiada więc programowi, ale jest obiektem (potencjalnie) aktywnym, któremu przydzielane są zasoby komputerowe, takie jak pamięć operacyjna oraz procesor. System operacyjny przekształcając program na proces tworzy pewne struktury danych. Służą one do przechowywania informacji o **atrybutach** procesu. Do atrybutów takich zalicza się m.in. opis zespołów przydzielonych danemu procesorowi oraz stan procesu: wykonywanie, gotowość lub zawieszenie. Mówi się, że proces jest **wykonywany,** jeżeli jest mu w danej chwili przydzielony procesor centralny. Jeżeli natomiast proces oczekuje wyłącznie na procesor, to jest on **gotowy.** Mówi się wówczas, że proces ten ma przydzielony **procesor wirtualny.** Jeżeli natomiast do kontynuacji procesu jest niezbędne zajście pewnego zdarzenia (np. zakończenie operacji wejścia), to jest on **zawieszony.**

Dwa procesy sekwencyjne są **współbieżne,** jeżeli wykonywanie jednego z nich zaczyna się po rozpoczęciu, ale przed zakończeniem wykonywania drugiego. Inaczej mówiąc, przedziały czasu, w których te procesy istnieją, nie są rozłączone. Procesy współbieżne mogą, ale nie muszą być wykonywane równolegle — zależy to zarówno od dostępnych środków technicznych, jak i od sposobu współpracy tych procesów ze sobą. Dlatego też mówi się często, że współbieżność jest **abstrakcją równoległości.** Powyższe określenie rozszerza się w naturalny sposób na układ więcej niż dwóch procesów. Mówi się więc, że mamy **układ procesów współbieżnych,** jeżeli każdy z nich jest współbieżny z jakimś (różnym od siebie) procesem z tego układu. Program, który specyfikuje układ procesów współbieżnych nazywa się **programem współbieżnym.**

Procesy współbieżne mogą operować rozłącznymi obiektami i w ogóle nie współpracować ze sobą. Mówi się wówczas o **procesach rozłącznych.** Jest to jednak sytuacja mało interesująca. Problemy pojawiają się dopiero wówczas, gdy procesy współbieżne są **interakcyjne,** tzn. albo konkurują między sobą w dostępie do zasobów, albo współpracują ze sobą komunikując się i wymieniając informacje. W obu tych wypadkach jest konieczna **synchronizacja procesów,** a więc czasowe wstrzymywanie jednego lub kilku z nich, dopóki nie zajdzie określone zdarzenie. Właśnie sprawa synchronizacji i komunikacji jest najtrudniejszym problemem do rozwiązania w programowaniu współbieżnym. Mo-

że się tu pojawić w szczególności bardzo niebezpieczna sytuacja — tzw. zakleszczenie (ang. deadlock). Proces jest w stanie zakleszczenia, jeżeli oczekuje na zdarzenie, które nigdy nie może nastąpić. Typowym przykładem zakleszczenia jest sytuacja, w której procesy **P1** i **P2** potrzebują zasobów **A** i **B**, a system operacyjny przydzielił już procesowi **P1** zasób **A**, zaś procesowi **P2** zasób **B**. Wówczas **P1** jest zawieszony nieskończenie w oczekiwaniu na przydział zasobu B, a P2 w oczekiwaniu na A. Unikanie zakleszczenia jest bardzo trudnym i ważnym zadaniem przy projektowaniu systemów operacyjnych, stanowiących układ procesów współbieżnych.

Synchronizacja

Jedną z najbardziej typowych reguł synchronizacyjnych jest **wzajemne wykluczanie** (ang. mutual exclusion). Jeżeli dwie operacje nie mogą być nigdy wykonywane jednocześnie, to mówi się, że muszą się one wzajemnie wykluczać w czasie. Konieczność wzajemnego wykluczania występuje bardzo często, np. w każdym wypadku, gdy jakiś proces musi mieć zagwarantowany wyłączny dostęp do wspólnego, dzielonego zasobu w celu wykonania na nim pewnych operacji. Przypuśćmy więc, że istnieje układ **n** procesów współbieżnych, z których każdy ma taką samą strukturę: w nieskończonej pętli na początku wykonuje pewne operacje nieistotne z punktu widzenia pozostałych procesów, a następnie przechodzi do operacji wymagających wyłączności. Tę pierwszą część nazywa się **operacjami lokalnymi**, a tę drugą — **rejonem krytycznym.** Instrukcje rejonu krytycznego muszą być ujęte w jakieś „ogrodzenie", gwarantujące spełnienie warunku wzajemnego wykluczania. Problem polega na zaproponowaniu mechanizmu oraz notacji dla tego właśnie „ogrodzenia". Dosyć szybko okazało się, że standardowe mechanizmy programowania sekwencyjnego mogą być użyte do tego celu, ale otrzymuje się wówczas rozwiązanie bardzo skomplikowane i podatne na błędy. Pojawiły się więc różne propozycje nowych konstrukcji, z których pewne zostały użyte praktycznie w projektowaniu i implementacji systemów operacyjnych. Poniżej przedstawiono krótko idee dwóch takich konstrukcji: **semaforów** oraz **monitorów.**

Semafory wprowadził Dijkstra ponad 20 lat temu. Był to pierwszy ogólnie zaakceptowany mechanizm synchronizacyjny. Na użytek niniejszego artykułu przyjęto następującą definicję, dość bliską oryginalnej:

Semafor jest to zmienna całkowita nieujemna, dla której są określone tylko trzy operacje: inicjowanie, **wait** oraz **signal.** Pierwsza z nich służy do nadania zmiennej semaforowej **s** wartości początkowej. Może być ona wykonana tylko jeden raz dla danej zmiennej i poza procesami. Dwie następne operacje są jednostkowe, niepodzielne i wzajemnie wykluczające się. Ich znaczenie jest następujące:

wait(s) — gdy **s** dodatnie, to $s := s - 1$, w przeciwnym razie proces jest zawieszany w oczekiwaniu na spełnienie tego warunku;

signal (s) — $s := s + 1$

Niech **mutex** będzie zmienną semaforową o wartości początkowej 1. Za pomocą tej zmiennej można w następujący sposób przedstawić schemat każdego z procesów z omawianego powyżej układu:

```
repeat
  operacje lokalne;
  wait (mutex);
  rejon krytyczny;
  signal (mutex)
until false
```

Nie wchodząc dalej w rozważania o semaforach wypada zauważyć, że jest to mechanizm niestrukturalny i w wypadku złożonego systemu procesów można względnie łatwo popełnić omyłkę niewykrywalną w czasie kompilacji programu współbieżnego, opisującego ten system. Dlatego też kilkanaście lat temu podjęto próby opracowania lepszego, strukturalnego mechanizmu synchronizacyjnego. W połowie lat siedemdziesiątych ukazały się prace proponujące nową konstrukcję, zwaną monitorem. Przyczynili się do tego trzej światowej sławy informatycy: Brinch Hansen, Dijkstra oraz Hoare. Pod pojęciem monitora rozumie się moduł grupujący zarówno struktury danych reprezentujące wspólne, dzielone przez procesy obiekty, jak i procedury opisujące jedyne wykonalne na tych zasobach operacje. Co więcej, procedury te z definicji wzajemnie wykluczają się w czasie. Tak więc monitor stanowi owo „ogrodzenie", w którym umieszcza się operacje rejonu krytycznego. Monitor poza swoim zasadniczym celem — synchronizacją dostępu do zmiennych dzielonych gwarantującą wzajemne wykluczanie — może także służyć do definiowania abstrakcyjnych typów danych oraz jako narzędzie modularyzacji programu. Zakres niniejszego artykułu nie pozwala niestety nawet na przytoczenie przykładu, gdyż należałoby wprowadzić jeszcze wiele pojęć, definicji oraz konwencji notacyjnych. Warto zauważyć, że monitor występuje jako konstrukcja w kilku językach programowania współbieżnego (np. Concurrent Pascal, Pascal+, Concurrent Euclid, Mesa, Chill i był użyty do implementacji niektórych systemów operacyjnych. Dotyczy to w szczególności wspomnianego w niniejszym artykule systemu Tunis, będącego strukturalną wersją Unixa zrealizowaną w języku Concurrent Euclid.

Współbieżność w systemie Unix

W systemie Unix każda czynność realizowana poza jądrem jest procesem. Jądro implementuje więc mechanizmy niezbędne do tworzenia i niszczenia procesów oraz do manipulowania nimi. Każdy istniejący proces może zażądać utworzenia nowego procesu, realizując operację jądra o nazwie **fork:**

processNo = fork ()

Jądro tworzy wówczas nowy proces, tzw. potomny, który jest prawie identyczny z macierzystym. Otrzymuje on oddzielną przestrzeń adresową, ale z tym samym programem, zmiennymi i z tymi samymi wartościami zmiennych. Jedyną różnicą jest wartość zmiennej **processNo** identyfikującej proces. W procesie macierzystym zmienna ta uzyskuje wartość niezerową, jednoznacznie identyfikującą proces potomny, natomiast w procesie potomnym jest ona zerowana. Oba procesy, jako procesy współbieżne, kontynuują realizację tego samego programu od instrukcji następującej po wywołaniu **fork.**

Zwykle jednak proces potomny tworzy się w celu wykonywania innego programu. Jest to możliwe dzięki operacji systemowej execve. W programie procesu powinna wystąpić wówczas instrukcja warunkowa, badająca wartość zmiennej **processNo** i w wypadku procesu potomnego wywołująca **execve** ze wskazaniem nowego programu, który ma zastąpić bieżąco wykonywany. Proces macierzysty z kolei może czekać na zakończenie procesu potomnego:

processNo = wait (status)

W wyniku powyższego wywołania operacji **wait** jądro wstrzymuje proces macierzysty aż do chwili, gdy zakończy się jeden z jego procesów potomnych. Identyfikator tego procesu potomnego jest przypisywany zmiennej **processNo,** zaś powód zakończenia jest opisany w słowie wskazywanym przez **status.** Każdy proces może się sam zniszczyć (zakończyć), wywołując operację **exit,** lub też zniszczyć inny proces operacją **kill.**

Współistnienie procesów można zilustrować na przykładzie procesu interpretującego komendy użytkownika (jest to proces systemowy nazywany **powłoką,** ang. shell). Proces ten w chwili odczytania komendy z klawiatury tworzy proces potomny, odpowiedzialny za realizację tej komendy. Standardowo proces macierzysty (czyli powłoka) oczekuje na zakończenie procesu potomnego i dopiero wówczas pobiera kolejną komendę. Jednakże jest możliwe wskazanie przez użytkownika, by dana komenda wykonywała się „w tle". Wówczas powłoka, nie czekając na zakończenie danego procesu potomnego, pobiera kolejną komendę i tworzy nowy proces (odpowiedzialny za tę nową komendę). Czynności te są realizowane z wykorzystaniem omówionych operacji **fork, execve, exit** oraz **wait.**

Procesy mogą się ze sobą komunikować. Najprostszym aparatem komunikacyjnym jest konstrukcja zwana **potokiem** (ang. pipe), będąca anonimowym, jednokierunkowym, ograniczonym **buforem.** Procesy połączone ze sobą takim buforem współpracują na zasadzie **producent-konsument.** Próba zapisania do pełnego bufora lub odczytania z pustego kończy się zawieszeniem procesu aż do chwili, gdy stan danego bufora odpowiednio się zmieni.

Normalny użytkownik siedzący przy terminalu nie korzysta w sposób jawny z omówionych powyżej operacji.

Forma podawania przez niego poleceń jest znacznie wygodniejsza — interpretator komend określa bowiem do tego celu pewien język wysokiego poziomu. Język ten jest bogaty i elastyczny, niemniej jednak przyjęty w nim zwyczaj stosowania zwięzłych skrótów bywa przedmiotem krytyki, zwłaszcza wśród nowicjuszy. W celu zilustrowania języka powłoki warto podać kilka prostych przykładów formułowania komend. Symbol „%" oznacza tu zgłoszenie się powłoki w oczekiwaniu na kolejną komendę.

Niech **prog** będzie nazwą pliku z programem, który należy wykonać. Wówczas zapisuje się to po prostu:

%prog

Przypuśćmy teraz, że **prog** oczekuje danych z klawiatury, a wyprowadza wyniki na ekran. Użytkownik natomiast żąda, aby dane były pobrane z pliku o nazwie **dane**, a nie z klawiatury. Można ten efekt uzyskać bardzo prosto, bez ingerencji w treść programu:

%prog<dane

Podobnie można zmienić miejsce wyprowadzania wyników, np. na plik o nazwie **wyniki:**

%prog>wyniki

Jest też możliwe połączenie tych obu żądań w jedną komendę:

%prog<dane>wyniki

Przypuśćmy teraz z kolei, że wyniki **prog** powinny być danymi wejściowymi dla programu **filtr**, który przykładowo zastępuje wielokrotne spacje pojedynczymi, a po tej operacji należy owe wyniki wydrukować na drukarce. Polecenie takie zapisuje się bardzo prosto:

%prog | filtr | lpr

Wówczas powłoka poleci utworzenie dwóch buforów do komunikacji pomiędzy procesami. Mamy wówczas do czynienia z **przetwarzaniem potokowym** (ang. pipeline). Program o nazwie **lpr** jest standardowym programem systemowym drukowania na drukarce.

Język interpretatora komend **(wersja Bourne shell i C shell)** ma wiele konstrukcji programistycznych spotykanych w normalnych językach programowania wysokiego poziomu, co pozwala na odejście od trybu interakcyjnego. Użytkownik może utworzyć program w tym języku, zapisać go na jakimś pliku, a następnie polecić powłoce wykonanie tego pliku. Stanowi to analogię do przetwarzania w trybie wsadowym.

Ceny ogłoszeń

Od 1 stycznia br. obowiązują następujące ceny ogłoszeń publikowanych na naszych łamach:
ogłoszenia duże (zależnie od objętości):
cała strona — 35 tys. zł, 3/4 — 30 tys., 1/2 — 25 tys., 1/4 — 20 tys., 1/8 — 15 tys.
ogłoszenia drobne (zależnie od liczby słów)
jedno słowo — 30 zł

Dodatki do ceny podstawowej:
— za dodatkowy kolor (na okładce) +30%
— za zamieszczenie ogłoszenia na czwartej stronie okładki +100%
— za zamieszczenie ogłoszenia na trzeciej stronie okładki +50%

Zniżki:

— za ogłoszenie 3—5-krotne —5%
— za ogłoszenia 6—10-krotne —10%
— za ogłoszenia 11-krotne i powyżej —20%
— za artykuły reklamowe i wkładki wykonane przez zleceniodawcę —40%
— za bloki i biuletyny wykonane przez zleceniodawcę — maks. —60%

W przypadku dostarczenia przez zleceniodawcę materiału ilustracyjnego nie odpowiadającego warunkom technicznym druku lub tekstu wymagającego redakcyjnego opracowania, do powyższych cen doliczane będą koszty odpowiednich usług fotograficznych, graficznych lub przygotowania tekstów.

Konferencje

SYSTEMS '87

W dniach od 19 do 23 października br. odbędzie się po raz dziesiąty międzynarodowa wystawa i kongres informatyczny SYSTEMS'87. Ta, odbywająca się co dwa lata, cykliczna impreza międzynarodowa zdobyła już znaczącą pozycję wśród czołowych europejskich wielkich imprez informatycznych. SYSTEMS od kilku lat współzawodniczy skutecznie ze słynną paryską wystawą SICOB, także powiązaną z międzynarodowym kongresem informatycznym (Convention Informatique). Współzawodnictwo obu imprez o europejski prymat wyrażało się w ubiegłych latach w maksymalnym zbliżaniu, a w jednym wypadku nawet w pokryciu się ich terminów, mimo że SYSTEMS odbywa się co dwa lata, a SICOB — co rok. Dopiero w bieżącym roku SICOB niespodziewanie przeniósł swój tradycyjny wrześniowy termin na początek kwietnia.

Organizatorzy na podstawie frekwencji 1985 roku twierdzą, że SYSTEMS'87 będzie największą europejską konferencją informatyczną (ok. 5000 uczestników). Rozmiary takie osiągnięto przez znaczną rozbudowę oraz zróżnicowanie struktury kongresu, który stanowi właściwie zespół wielu konferencji.

Imprezą podstawową będzie doroczny kongres zachodnioniemieckiego stowarzyszenia informatycznego GI (Gesellschaft für Informatik), w tym roku pod bardzo aktualnym hasłem „Systemy baz wiedzy".

Drugą imprezą konferencyjną będą seminaria adresowane do użytkowników, organizowane pod patronatem branżowych stowarzyszeń i instytucji. Seminaria te dotyczyć będą ośmiu dziedzin zastosowań: budownictwa, handlu, przemysłu, bankowości, medycyny, administracji publicznej, dystrybucji leków oraz ubezpieczeń.

Trzecią grupą obrad będą seminaria adresowane do specjalistów, ukierunkowane na najważniejsze aktualne problemy informatyki, takie jak: standaryzacja, inżynieria oprogramowania, bazy danych, sieci lokalne, przetwarzanie w czasie rzeczywistym, rozpoznawanie obrazów, nowe technologie zapisu i przesyłania danych itp. Wydaje się, że taka organizacja obrad znacznie podnosi ich efektywność i dlatego może być wzorem również dla organizatorów naszych konferencji informatycznych.

W tym roku po raz pierwszy z programu konferencyjnego SYSTEMS wyłączono cały dział zastosowań inżynierskich typu CA (Computer Assisted), które objęła nowa cykliczna impreza SYSTEC, zainicjowana w Monachium po raz pierwszy w roku ubiegłym przez tego samego organizatora tzn. Towarzystwo Targów i Wystaw Monachijskich (Münchener Messe- und Ausstellungsgesellschaft), a przewidziana jako przemienna z SYSTEMS, tzn. co dwa lata.

Konferencji i seminariom tradycyjnie będzie towarzyszyć rosnąca pod względem powierzchni (105 000 m² w 1985 r.), liczba wystawców (1146 w 1985 r.) oraz liczby zwiedzających (133 000 w 1985 r.) międzynarodowa ekspozycja sprzętu komputerowego, oprogramowania, materiałów pomocniczych oraz wydawnictw, co szybko zbliża ją do rozmiarów hanowerskiej CeBIT oraz paryskiej SICOB.

Rozmieszczoną w kilkunastu halach ekspozycję podzielono na osiem następujących obszarów tematycznych:

- oprogramowanie i usługi informatyczne,
- moduły systemów komputerowych (urządzenia peryferyjne, podzespoły OEM),
- telekomunikacja,
- systemy biurowe,
- wyposażenie pomocnicze i akcesoria,
- centrum informacyjne (informacje dla zwiedzających, stowarzyszenia, wydawnictwa, usługi informacyjne z baz danych),
- pierwsze kroki w przetwarzaniu informacji,
- systemy komputerowe dla przedsiębiorstw oraz instytucji administracji publicznej.

ROBERT KOWALSKI
Imperial College of Science and Technology
Uniwersytet Londyński
Wielka Brytania

Logika w systemach ekspertowych (2)

W drugiej części artykułu zilustrowano sposób rozwiązywania problemów charakterystycznych dla systemów ekspertowych na przykładzie rozwiązywania równań matematycznych.

ROZWIĄZYWANIE RÓWNAŃ METODĄ SYSTEMU EKSPERTOWEGO

Poniższy przykład jest rekonstrukcją prologowego systemu PRESS [2] rozwiązującego układy równań. W celu skrócenia opisu pominięto reguły upraszczania podwyrażeń takich jak (4—2) i (6/3) oraz użyto nieformalnej składni w rodzaju 2t zamiast 2*t. Proces rozwiązywania ma strukturę drzewa. Na najwyższym poziomie ogólności pojawiają się tylko dwa kroki:

DANE
(4t + 3) — 2t = 6

KROK 1
Zredukuj liczbę wystąpień t do jednego.
(4 — 2)t + 3 = 6

KROK 2
Izoluj wystąpienie t.
t = (6 — 3) / (4 — 2)

CEL

Proces ten można sformalizować za pomocą reguły:

Rozwiązane (u x v)
jeśli Zredukowane (u x u')
i Izolowane (u' x v)

co w zapisie proceduralnym można wyrazić tak:

Aby rozwiązać równanie u ze względu na zmienną x (otrzymując w wyniku równanie v), należy zredukować liczbę wystąpień x w równaniu u do jednego (otrzymując w wyniku równanie u') oraz wykonać izolację zmiennej x w równaniu u' (otrzymując w wyniku równanie v).
Krok 1 można rozłożyć na dwa kroki:

DANE
(4t + 3) — 2t = 6

KROK 1.1
Zgrupuj wystąpienia t.
(4t — 2t) + 3 = 6

KROK 1.2
Połącz sąsiednie wystąpienia t.
(4 — 2)t + 3 = 6

Krok 1 jest stosunkowo prosty, ponieważ są tylko dwa wystąpienia t. W ogólnym wypadku jednak jest niezbędna wielokrotna redukcja par wystąpień zmiennej. Formalnie można to zapisać za pomocą dwóch reguł:

Zredukowane (u x v)
jeśli ZredukowanaPara (u x u')
i Zredukowane (u' x v)

Zredukowane (u x u)
jeśli Nie ZredukowanaPara (u x u')

Powyższe reguły kroku 1 należy uzupełnić o następujące dwie reguły:

ZredukowanaPara (u x v)
jeśli Zgrupowane (u x u')
i ZredukowanaPara (u' x v)
ZredukowanaPara (u x v)
jeśli Połączone (u x v)

Krok 2 składa się również z dwóch kroków, które służą do kolejnego przenoszenia podwyrażeń z lewej strony na prawą stronę równania, aż do pełnej izolacji zmiennej:
DANE
(4 — 2)t + 3 = 6

KROK 2.1
Odejmij 3 od obu stron i skróć podwyrażenia po lewej stronie.
(4 — 2)t = (6 — 3)

KROK 2.2
Podziel obie strony przez (4 — 2) i skróć podwyrażenia po lewej stronie.
t = (6 — 3) / (4 — 2)

CEL

Działania te można sformalizować za pomocą następujących reguł:

Izolowane ((u1 s u2 = u3) x v)
jeśli Nie x w wyrażeniu u2
i Odwrotność (s s')
i Izolowane ((u1 = (u3 s' u2)) x v)
Izolowane ((x = y) x (x = y))

Odwrotność (+ —)

Odwrotność (* /)
itd.

Zauważmy, że równania postaci

wyrażenie1 = wyrażenie2

są reprezentowane przez listy

(wyrażenie1 = wyrażenie2)

gdzie „=" jest stałą a nie symbolem predykatu.

W celu uzupełnienia omawianego przykładu należy jeszcze (poza zdefiniowaniem predykatów niższego poziomu takich jak „w wyrażeniu") podać reguły definiujące operacje grupowania i łączenia zmiennych:

Połączone (u x v)
jeśli ((u1*u2) + (u3*u2)) w wyrażeniu u
i x w wyrażeniu u2
i zamiana ((u1*u2) + (u3*u2)) na ((u1 + u3)*u2) w wyrażeniu u daje v

Zgrupowane (u x v)
jeśli ((u1 + u2) + u3) w wyrażeniu u
i x w wyrażeniu u1
i x w wyrażeniu u3
i zamiana ((u1 + u2) + u3) na ((u1 + u3) + u2) w wyrażeniu u daje v

Reguła łączenia stwierdza, że prawo rozdzielności

((u1*u2) + (u3*u2)) = (u1 + u3)*u2

może być zastosowane „wstecz" w celu połączenia wystąpień x w wyrażeniu u2; reguła grupowania mówi, że twierdzenie

((u1 + u2) + u3) = ((u1 + u3) + u2)

może być wykorzystane do zgrupowania wystąpień x w wyrażeniu u1 oraz u3. Twierdzenie to wynika z praw przemienności i łączności

(x + y) = (y + x)
((x + y) + z) = (x + (y + z))

PODEJŚCIE HEURYSTYCZNE I ALGORYTMICZNE

Omówiony przykład ilustruje przyrostowy sposób budowy systemu ekspertowego przez dodawanie kolejnych reguł. Aby rozwiązać inną klasę zadań, np. równanie

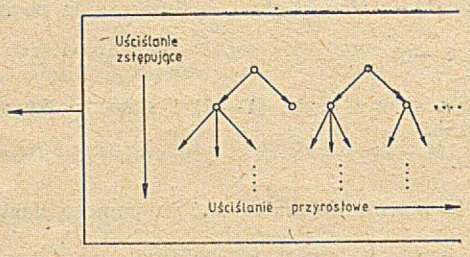
$t^2 + 3t + 2 = 0$

ze względu na t, należy uzupełnić istniejące reguły nowymi regułami. Poszczególne reguły lub zbiory reguł mają charakter heurystyczny. Chociaż umożliwiają roz-

wiązanie pewnych zadań, jednak nie gwarantują rozwiązania wszystkich. Pojawiają się więc tutaj elementy niedeterminizmu i przeszukiwania przestrzeni rozwiązań, co jest podyktowane koniecznością znalezienia reguł odpowiednich dla danego zadania.

Gdy zbiór reguł heurystycznych obejmuje wszystkie przypadki z określonej klasy zadań, staje się on w rzeczywistości algorytmem dla danej klasy zadań. W pewnym sensie więc podejścia heurystyczne i algorytmiczne uzupełniają się: pełen zestaw heurystyk tworzy algorytm.

Dodawanie nowych reguł może posłużyć jako przykład programowania metodą kolejnych uściśleń. Programowanie przez dodawanie reguł stanowi więc uzupełnienie zstępującej metody kolejnych uściśleń, polegającej na definiowaniu ogólnych pojęć i stopniowym schodzeniu do pojęć coraz niższego poziomu, aż do osiągnięcia poziomu języka programowania (tak jak w przykładzie



Ilustracja sposobów programowania

zadania sortowania). Te dwa sposoby programowania można przedstawić graficznie, jak na rysunku, gdzie lokalnemu rozgałęzieniu węzła (w drzewie „i-lub") odpowiada wykonanie redukcji zadania w zależności od warunków i wniosków danej reguły.

Języki programowania logicznego ułatwiają stosowanie w praktyce obu metod programowania zstępującego. Jest to spowodowane świadomą rezygnacją z pojęcia zmiennej globalnej oraz instrukcji przypisania, a co za tym idzie względną niezależnością poszczególnych reguł. Większość konwencjonalnych technik budowy oprogramowania jest oparta również na zstępującej metodzie uściśleń. Programowanie metodą przyrostowych uściśleń jest analogiczne do metody Dijkstry z użyciem rozkazów dozorowanych [8].

METODA PRÓB I BŁĘDÓW A PODEJŚCIE ŚCISŁE

Technika systemów ekspertowych umożliwia również inny sposób wykorzystania metody kolejnych uściśleń, a mianowicie programowanie metodą prób i błędów. Metoda ta jest zbliżona do konwencjonalnego procesu testowania i poprawiania programów. Jest szczególnie przydatna w systemach ekspertowych, częściowo dlatego, że zapewnia naturalny sposób gromadzenia wiedzy ekspertów, a także dlatego, że umożliwia łatwą modyfikację reguł.

Najprostsza modyfikacja polega na dodaniu jednego lub kilku warunków do ogólnej reguły. Ma to oczywiście na celu ograniczenie zastosowania reguły, a tym samym uniknięcie wyprowadzania błędnych wniosków. Modyfikacja taka jest na przykład konieczna w pierwszej regule izolacji zmiennej w celu uniknięcia dzielenia przez zero.

Bardziej skomplikowaną modyfikacją jest dodanie nowych parametrów do predykatu. Na przykład „sprzeczność"

Wielkość (Dumbo duży)
Wielkość (Dumbo mały)

można usunąć przez dodanie trzeciego parametru w roli niezbędnego punktu odniesienia

Wielkość (Dumbo mały słoń)
Wielkość (Dumbo duży ssak)

W metodzie prób i błędów wyróżnia się dwa rodzaje błędów: błędy zauważalne (ang. errors of commission,) które są w rzeczywistości niepoprawnymi lub niedopuszczalnymi wynikami przeprowadzonych prób, oraz błędy niezauważalne (ang. errors of ommission), które nie są żadnymi wynikami, gdyż przeprowadzone próby nie wywołują zauważalnych zmian. Błędy niezauważalne można poprawić przez przyrostowe dodawanie nowych reguł, błędy zauważalne — przez modyfikację istniejących reguł. E. Shapiro [13] skonstruował elegancki algorytm wykrywający obydwa rodzaje błędów w programach logicznych. Elegancja i zarazem praktyczność tego algorytmu wynika z wykorzystania aparatu logiki oraz z braku efektów ubocznych.

Jednak poprawianie programów metodą prób i błędów jest na ogół uważane za złą praktykę. W szczególności konstruowanie systemów ekspertowych tą metodą dość często spotyka się z ostrą krytyką.

Ścisłe podejście do budowy oprogramowania ma na celu tworzenie programów zgodnie z ich specyfikacjami. Należy tu uwzględnić dwa aspekty — poprawność i zupełność. Program jest poprawny, jeśli każdy wynik generowany przez program pozostaje w zgodzie ze specyfikacją. Program jest zupełny, jeśli każdy wynik dopuszczalny z punktu widzenia specyfikacji może być również wygenerowany przez program.

W sytuacjach, w których zbadanie zupełności i (lub) poprawności jest istotne, a skonstruowanie formalnej specyfikacji możliwe, przeprowadzenie niezbędnych dowodów dla systemów regułowych wydaje się być zadaniem co najmniej tak łatwym jak dla programów konwencjonalnych. Jeśli w szczególności użyje się aparatu logiki zarówno do budowy specyfikacji, jak i programów, to zadanie ulegnie istotnemu uproszczeniu. Na przykład, poprawność programu regułowego można na ogół dowieść przez wykazanie poprawności każdej reguły z osobna. Metoda ta została rozwinięta [4, 5, 9—11] jako rozszerzenie technik transformacji programów Darlingtona i Burstalla [3]. Nie stosuje się jej jednak do takich formalizmów systemów ekspertowych, które dopuszczają występowanie instrukcji podstawienia i innych efektów ubocznych.

Badanie zupełności programów regułowych jest zadaniem znacznie bardziej skomplikowanym, ale z pewnością nie trudniejszym niż w wypadku oprogramowania konwencjonalnego. Tak więc, o ile reguły są wyrażone w systemie logiki bez efektów ubocznych, to systemy ekspertowe w porównaniu z programami konwencjonalnymi charakteryzują się łatwiejszym dowodzeniem poprawności i nie trudniejszym dowodzeniem zupełności.

Mimo to, w dziedzinie budowy systemów ekspertowych nadal pojawiają się kłopotliwe pytania co do celowości, a nawet możliwości konstruowania formalnych specyfikacji zadań. Pytania takie nie pojawiają się oczywiście w kontekście prostych przykładów podanych w artykule. Jednak, na przykład, w dziedzinie diagnostyki medycznej można wątpić, czy zupełne specyfikacje formalne są w ogóle możliwe. W takich wypadkach trzeba zadowolić się nieformalnym kryterium „system działa przynajmniej tak dobrze jak ekspert". Okazać się więc może, że jedyną drogą do realizacji takich systemów ekspertowych będzie metoda prób i błędów.

Programowanie metodą prób i błędów może być przydatne nawet wówczas, gdy skonstruowanie specyfikacji formalnych jest możliwe i celowe. Przykładem może być zadanie rozwiązywania równań. Reguły algebry stanowią specyfikację programu, a system ekspertowy jest programem. Konstruowanie systemu ekspertowego metodą prób i błędów można uznać za rozszerzenie konwencjonalnego programowania metodą kolejnych uściśleń. Błędów niezauważalnych można pozbyć się dodając nowe reguły i zwiększając tym samym zupełność programu. Błędy zauważalne można poprawić modyfikując istniejące reguły i zwiększając tym samym poprawność programu. Co więcej, można tutaj zastosować wspomaganie komputerowe z wykorzystaniem wykonywalnych specyfikacji [12] sformułowanych za pomocą aparatu logiki (tak jak np. w algorytmie podanym przez Shapiro).

Tak więc logika daje możliwość pogodzenia empirycznego podejścia opartego na metodzie prób i błędów z podejściem ścisłym opartym na specyfikacji i weryfikacji programu.

NIEDOKŁADNOŚĆ

Niedokładność w systemach ekspertowych przyjmuje wiele różnych postaci. Na przykład, programowanie metodą prób i błędów można potraktować jako pewną formę niedokładności: reguły heurystyczne mogą być jedynie częściowo poprawne na poszczególnych etapach budowy programu.

Niedokładność można określać jawnie przez przypisanie poszczególnym relacjom prawdopodobieństwa lub innej miary niepewności. Na przykład, aby sformalizować stwierdzenie:

prawdopodobieństwo, że ubytek powietrza w oponie jest spowodowany nieszczelnym wentylem, wynosi 60%

można wprowadzić dodatkowy parametr określający prawdopodobieństwo przyczyny niesprawności, np.:

powoduje (nieszczelny-wentyl, ubytek-powietrza 0.6).

Aby reguły tego rodzaju można układać w łańcuchy przyczynowe, należy wprowadzić odpowiedni mechanizm wyznaczania prawdopodobieństwa wynikowego:

Prowadzi-do (x y c)
jeśli Powoduje (x y c)
Prowadzi-do (x y c3)
jeśli Powoduje (x z c1)
i Prowadzi-do (z y c2)
i Łączy (c1 c2 c3)

Pozostaje jeszcze odpowiednio zdefiniować relację „Łączy", co zresztą nie jest konieczne na tym etapie budowy programu [1]). Mechanizm wyliczania prawdopodobieństwa relacji został wykorzystany w pracach [6, 7].

Alternatywnym podejściem może być tworzenie nowych systemów logiki opartych na pojęciu prawdopodobieństwa lub na pojęciu zbiorów rozmytych. Zalecana przez autora metoda formalizacji niepewności w ramach klasycznego systemu logiki „nierozmytej" ma jednak dwie istotne zalety:

(1) w celu wprowadzenia pojęcia niepewności nie trzeba zmieniać istniejącego systemu logiki, a więc ani jego semantyki, ani teorii dowodzenia;

(2) definiowanie nowego pojęcia następuje dopiero w relacjach niższego poziomu (np. w relacji „Łączy"); różne definicje niższego poziomu mogą być dodawane bez zmiany poziomów, co daje większą modularność i elastyczność.

Te same argumenty znajdują oczywiście zastosowanie w wypadku wprowadzania do klasycznego systemu logiki innych pojęć, np. w wypadku konieczności uwzględnienia czasu.

Jeszcze innym podejściem jest potraktowanie niepewności jako pewnego pojęcia metalogicznego, określającego prawdopodobieństwo tego, że konkretna reguła lub asercja może znaleźć zastosowanie w danej sytuacji. Niech, na przykład, „nierozmytym" celem będzie znalezienie takiego x, że x powoduje ubytek powietrza w oponie, tzn.

Powoduje (x ubytek-powietrza).

Załóżmy również, że dane są dwie „nierozmyte" asercje

Powoduje (nieszczelny-wentyl ubytek-powietrza)
Powoduje (przebita-dętka ubytek-powietrza)

Analiza statystyczna dotychczasowych prób rozwiązywania podobnych zadań mogłaby wykazać, że pierwsza asercja była prawdziwa w 60% badanych wypadków, natomiast druga asercja w 39%. Taka informacja mogłaby być użyteczna w relacjach metapoziomu w celu wybrania reguły lub asercji, która była w przeszłości prawdziwa dla większej liczby wypadków. Wykonywanie dedukcji na poziomie podstawowym oraz na metapoziomie [1] jest istotnym rozszerzeniem klasycznej logiki pierwszego rzędu.

* * *

Wszystkie przykłady podane w tym artykule są programami logicznymi sformułowanymi za pomocą Prologu. Obejmują one kilka specyfikacji programów, kilka programów algorytmicznych oraz jeden system ekspertowy. Jednocześnie wszystkie te przykłady są systemami regułowymi. Stanowi to dowód na to, że systemy regułowe nie mogą być utożsamiane z systemami ekspertowymi.

Wiele często cytowanych zalet systemów ekspertowych, takich jak zrozumiałość, modyfikalność i umiejętność uzasadniania wyprowadzonych wniosków, to w rzeczywistości zalety języków regułowych, które posłużyły do implementacji systemów. Zalety te nie zależą od rodzaju zastosowania i dotyczą w równej mierze specyfikacji programów jak i programów algorytmicznych zrealizowanych w językach regułowych.

Nie wszystkie języki regułowe są sobie równe. Logika jest jedynym językiem regułowym, który może być zastosowany ni tylko do budowy systemów ekspertowych, lecz także do specyfikacji programów i pisania programów algorytmicznych. Co więcej, nawet w wypadku systemów ekspertowych takie ich cechy jak zrozumiałość, modyfikowalność i umiejętność uzasadniania podjętych akcji, są osiągalne jedynie w takim stopniu, w jakim reguły mogą zachować swą względną niezależność. Wykorzystanie instrukcji przypisania i innych efektów ubocznych poważnie osłabia te cechy. Programowanie logiczne oraz czysty Prolog nie dopuszczają efektów ubocznych (w przeciwieństwie do Prologu „bez ograniczeń" i innych systemów regułowych).

Zalety języków regułowych nie stanowią jednak jedynych zalet systemów ekspertowych. Dzięki empirycznemu podejściu opartemu na metodzie prób i błędów systemy ekspertowe mogą być użyte w nowych dziedzinach zastosowań. O ile podejście algorytmiczne zakłada znalezienie pełnego i ścisłego rozwiązania danego zadania, o tyle podejście przyjęte w technikach systemów ekspertowych zadowala się zbiorem heurystyk, które są często niepełne i tylko w przybliżeniu poprawne. System ekspertowy może być oczywiście oceniany przez porównanie jego działania z formalną specyfikacją. Może być również oceniany przez porównanie z dostępną wiedzą ludzką. W ten sposób można ocenić nie tylko ogólne działanie systemu, lecz także wiedzę zawartą w poszczególnych heurystykach.

Ta wiedza właśnie jest najważniejszą częścią systemu ekspertowego. Teza tego artykułu brzmi: logika stanowi najlepszy formalizm, w ramach którego wiedza może być najpierw wysłowiona, a następnie wykorzystana.

Prace te były wykonane przy wsparciu brytyjskiego urzędu Science and Engineering Research Council. Przykład gry w piętnaście pochodzi od Maartena van Emden, a wiele pomocnych uwag do pierwszych wersji tego artykułu wnieśli Alan Bundy, John Fox i Bob Welham.

LITERATURA

[1] Bowen K., Kowalski R. A.: Amalgamating language and metalanguage in logic ,programming. Pp. 153—173, Clark K. L., Tarnlund S. A. Eds., Logic Programming, Academic Press, London, 1982.

[2] Bundy A., Welham R.: Using Meta-level Inference for Selective Application of Multiple Rewrite Rules in Algebraic Manipulation. Pp. 24—38, Proc. 5th Conference on Automated Deduction, Springer-Verlag, Berlin, 1980.

[3] Burstall R. M., Darlington J.: A transformation system for developing recursive programmes. J. of ACM, Vol. 24, No. 1, pp. 44—67, 1975.

[4] Clark K. L., Darlington J.: Algorithm classification through synthesis. Computer Journal, Vol. 23, No. 1, 1980.

[5] Clark K. L.: Synthesis and Verification of Logic Programs. Report DOC 81/36, Imperial College, London, 1981.

[6] Clark K. L., McCabe F. G.: PROLOG — A Language for Implementing Expert Systems. Machine Intelligence, Vol. 10, 1982.

[7] Clark K. L., McCabe F. G.: Micro-PROLOG-Programming in Logic. Prentice-Hall, Englewood Cliffs (NJ), 1983.

[8] Dijkstra E. W.: A Discipline of Programming. Prentice-Hall, Englewood Cliffs (NJ), 1976.

[9] Hogger C. J.: Goal-Oriented Derivation of Logic Programs. Lecture Notes in Computer Science, Vol. 64, Springer-Verlag, 1978.

[10] Hogger C. J.: Derivation of Logic Programs. JACM 28, No. 2, pp. 372—392, 1981.

[11] Hogger C. J.: Introduction to Logic Programming, (A.P.I.C.S. Series), to be published by Academic Press Inc., 1983.

[12] Kowalski R. A.: The Relationship between Logic Programming and Logic Specification. Proceedings of the Specification Workshop, York University, 1983.

[13] Shapiro E.: Algorithmic Programming Debugging, MIT Press, 1982.

[1]) Tego rodzaju opóźnienie definiowania pewnych relacji jest typowe dla programowania logicznego.

ZBIGNIEW SZKARADNIK
Instytut Informatyki
Politechnika Śląska
Gliwice

Grafika żółwia w języku Forth (2)

W poprzedniej części artykułu [1] krótko omówiono sposób realizacji w języku Forth procedur tzw. grafiki żółwia. Przedstawiony tam pakiet programowy może stanowić zalążek języka graficznego (rys. 1) zbudowanego na bazie Fortha, pod warunkiem odpowiedniej jego rozbudowy (dobudowanie obsługi błędów, dołączenie większej liczby graficznych urządzeń we-wy itp.).

Jednakże nawet tak skromny zestaw słów umożliwia konstruowanie prostych programów. Są to oczywiście przykłady figur bardzo prostych, które jednak same mogą stanowić „cegiełki", służące następnie do składania bardziej skomplikowanych wzorów.

Proste figury

W pierwszej części artykułu przedstawiono wydruk programu rysującego trójkąt równoboczny. Program ten można łatwo rozbudować, tak aby powodował wykreślenie dowolnego wielokąta foremnego. Wystarczy zmodyfikować jedynie liczbę boków i kąt (wydruk 1).

```
0 var n

: polygon          ( edge n --- )
  dup -> n
  1+ 0 do
     dup forward 360 n / left ( or right )
  loop drop ;
```

Wydruk 1. Program wykreślający wielokąt

Zauważmy, że przy dużym n (na przykład, równym 360) obraz wielokąta kreślonego w ten sposób będzie okręgiem (dla n nie będących podzielnikami 360 pojawią się błędy). Pakiet grafiki żółwia gwarantuje także inne sposoby wykreślania okręgu. Można to zrobić wykorzystując zdefiniowane wcześniej słowo (circle). Podobny efekt można uzyskać wykreślając łuk (arc) o 360 (lub —360) stopni. W tym wypadku jednak punkt początkowy (przed rozpoczęciem kreślenia) będzie znajdował się na obwodzie okręgu, a nie w jego środku. Można wreszcie zdefiniować okrąg w sposób rekurencyjny (wydruk 2).

```
: r_circle          ( n --- )
  dup forward 1 left ( or right ) ?break recursive r_circle ;

10 r_circle         ( kresli rekurencyjnie okrag o obwodzie 3600 )
```

Wydruk 2. Program wykreślający rekurencyjnie okrąg

We wszystkich trzech metodach okrąg jest rysowany w kierunku przeciwnym do ruchu wskazówek zegara. W ostatnich dwóch wariantach można łatwo uzyskać przeciwny kierunek kreślenia.

Figury złożone

Podane przykłady są oczywiście bardzo proste. Można jednak wykreślić coś bardziej skomplikowanego, np. figurę z okładki numeru 5'1985 „Informatyki". Można ją wykreślić przy użyciu programu słowa {rosette} zamieszczonego na wydruku 3. Efekt otrzymany dla kąta równego 92 stopnie przedstawiono na rys. 2.

Również dość ciekawe efekty można uzyskać w bardzo prosty sposób, rysując kolejno elipsy o zmieniających się osiach za pomocą programu słowa {pillow} (wydruk 3, rys. 3). W podobny sposób uzyskana została figura z rysunku 1.

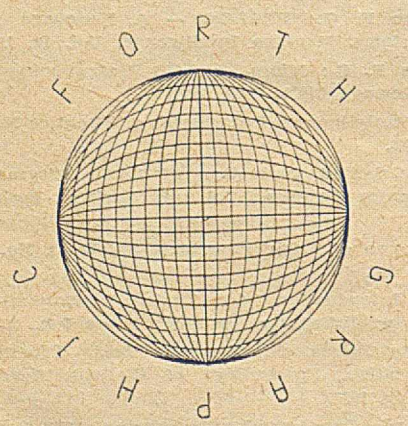
```
Screen : 1
  0 ( rosette )
  1
  2 0 var alfa     0 var size      0 var n
  3 0 var dr       0 var r
  4
  5 : rosette ( alfa n size --- )
  6   -> size      -> n       -> alfa
  7   size n 2* / dup -> dr -> r
  8   n 0 do
  9     alfa left r forward   r dr + -> r   ?break
 10   loop ;
 11
 12 : pillow -          ( size --- )
 13   2/ dup 1+ 0 do
 14             dup i - i ellipse ?break
 15           dup 10/ +loop drop ;
```

Wydruk 3. Słowa wykreślające figury złożone

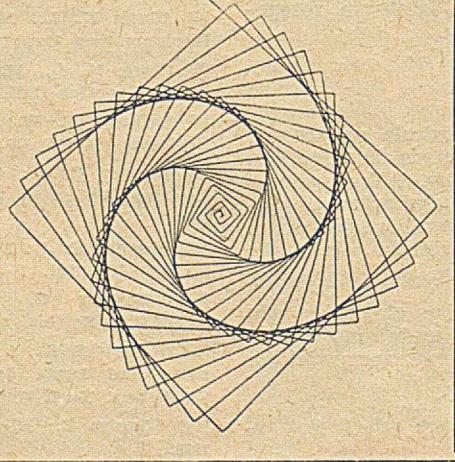
Krzywe rekurencyjne

Osobną grupę ciekawych figur można uzyskać kreśląc krzywe rekurencyjne. Program wykreślający rekurencyjne krzywe Sierpińskiego (rys. 4) zamieszczono na wydruku 4. Często pokazywany w telewizji „płatek śniegu" (rys. 5) można otrzymać przy użyciu programu słowa {flake} z wydruku 4. Zauważmy, że nieznaczna modyfikacja tego programu (zmiana {left} na {right} i odwrotnie w słowie {p}) prowadzi do uzyskania zupełnie nowej figury. Składając w ten sposób otrzymane figury można łatwo uzyskać „serwetę" z fotografii 1.

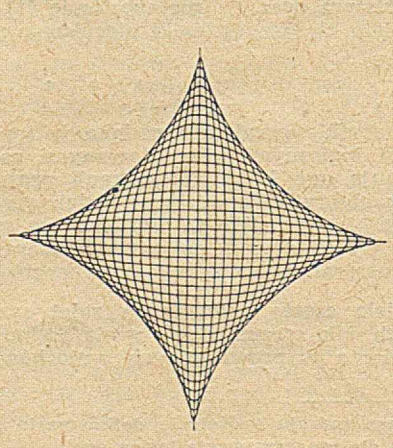
Rys. 1. Ilustracja grafiki w języku Forth



Rys. 2. Użycie słowa {rosette}



Rys. 3. Użycie słowa {pillow}



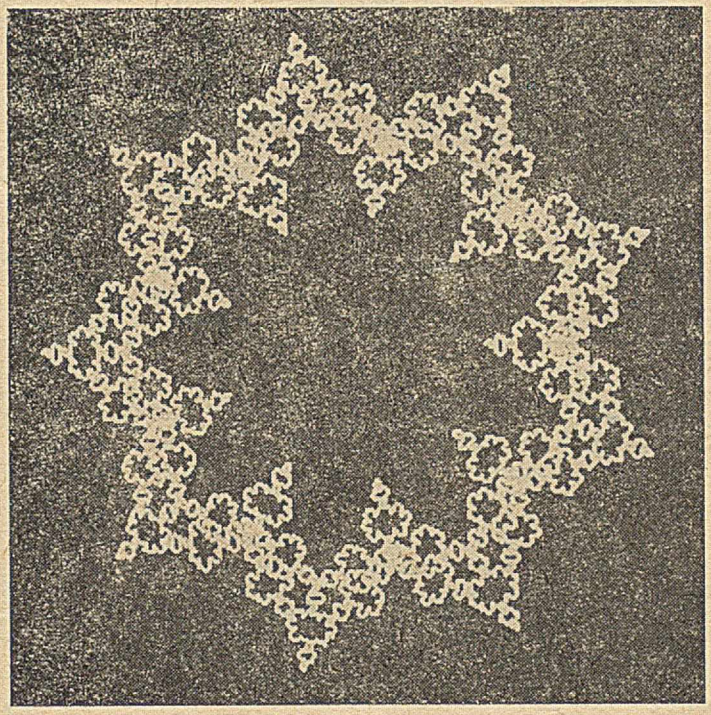
```
Screen # 2                                          File : EXAMPLE.GRF
  0 ( Sierpinski curves )
  1
  2 0 var edge    0 var n    0 var diagonal
  3
  4 : s ( n --- )
  5   ?break dup 0= if drop
  6          else dup 1- recursive s  45 right diagonal forward 45 right
  7               dup 1- recursive s  90 left  edge  forward 90 left
  8               dup 1- recursive s  45 right diagonal forward 45 right
  9                    1- recursive s
 10          endif ;
 11
 12 : Sierpinski       ( edge n --- )
 13   -> n -> edge
 14   edge 100 m* 141 m/ -> diagonal edge move
 15   4 0 do n s 45 right diagonal forward 45 right loop ;
```

```
Screen # 3                                          File : EXAMPLE.GRF
  0 ( snow flake )
  1
  2 : p                 ( edge n --- )
  3   ?break dup 0= if drop forward
  4          else 2dup 1- swap 3 / swap recursive p  60  left
  5               2dup 1- swap 3 / swap recursive p  120 right
  6               2dup 1- swap 3 / swap recursive p  60  left
  7                    1- swap 3 / swap recursive p
  8          endif ;
  9
 10 : flake             ( edge n --- )
 11   swap dup 5773 m* 10000 m/ swap drop dup move rot rot swap
 12   150 right 3 0 do
 13     2dup p 120 right
 14   loop drop drop
 15   -- right move 180 right ;
```

Wydruk 4. Słowa wykreślające figury rekurencyjne

Animacja obrazu

Dysponując monitorem graficznym (przy użyciu którego można zrealizować słowa usuwające narysowaną linię) można pokusić się o próbę animacji rysunku. Najprostszym tego przykładem jest zegar z ruchomymi wskazówkami



Fot. 1. Serwetka otrzymana ze złożenia figur

```
Screen # 4                                          File : EXAMPLE.GRF
  0 ( clock )
  1
  2 0 var angle       0 var hand
  3
  4 : dash              ( r --- )
  5   dup 9 * 10 / move dup 10 / forward minus move ;
  6
  7 : point             ( r --- )
  8   dup move x y draw minus move ;
  9
 10 : dial              ( r --- )
 11   direction -> angle dup -> hand
 12   60 0 do
 13        dup i 5 mod 0= if dash else point endif
 14        -6 turn
 15      loop drop ; -->
```

```
Screen # 5                                          File : EXAMPLE.GRF
  0 ( clock )
  1 : mark  hand dup
  2   angle 2* -> direction dash
  3   angle 2+ -> direction dash ;
  4
  5 : sec_hand          ( sec --- )
  6   6 * minus angle + -> direction
  7   hand 9 * 10 / dup move backward ;
  8
  9 : min_hand          ( min --- )
 10   6 * minus angle + -> direction
 11   hand 4 * 5 / dup move backward ;
 12
 13 : hrs_hand          ( hrs --- )
 14   30 * minus angle + -> direction
 15   hand 3 * 5 / dup move backward ; -->
```

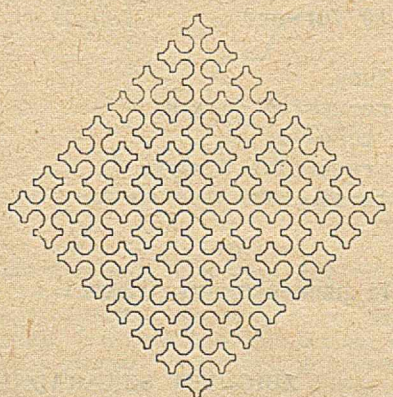
```
Screen # 6                                          File : EXAMPLE.GRF
  0 ( clock )
  1
  2 0 var oldsec   0 var oldmin   0 var oldhrs
  3
  4 : hands
  5   time dup -> oldhrs hrs_hand
  6        dup -> oldmin min_hand
  7        dup -> oldsec sec_hand ;
  8
  9
 10
 11 0  var  sec  0  var  min  0 var hrs
 12
 13 -->
 14
 15
```

```
Screen # 7                                          File : EXAMPLE.GRF
  0 ( clock )
  1
  2 : timer
  3   begin ?break
  4   time -> hrs -> min -> sec
  5   oldsec sec = if else
  6     oldsec sum_down sec_hand
  7     oldsec oldmin = if pen_down oldsec min_hand endif
  8     oldsec oldhrs 5 * = if pen_down oldhrs hrs_hand endif
  9     sec pen_down sec_hand sec -> oldsec
 10     sec 0= if oldmin sum_down min_hand
 11          oldmin sec = if pen_down oldmin sec_hand endif
 12          oldmin oldhrs 5 * = if pen_down oldmin hrs_hand endif
 13          min pen_down min_hand min -> oldmin
 14          min 0= if oldhrs sum_down hrs_hand
 15               oldhrs 5 * sec = if pen_down sec sec_hand endif -->
```

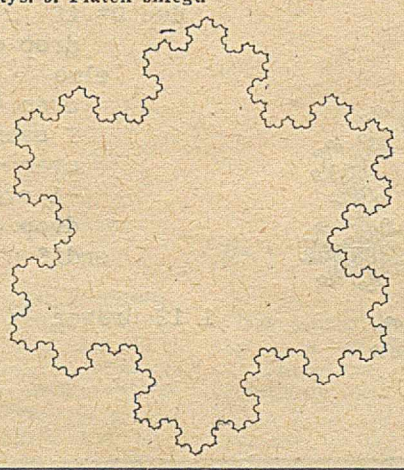
```
Screen # 8                                          File : EXAMPLE.GRF
  0 ( clock )
  1
  2            oldhrs 5 * min = if pen_down min min_hand endif
  3        hrs pen_down hrs_hand hrs -> oldhrs
  4      endif endif endif
  5    again ;
  6
  7 : clock             ( r --- )
  8   dial mark hands timer ;
  9
 10
 11
 12
 13
 14
 15
```

Wydruk 5. Program wykreślający zegar

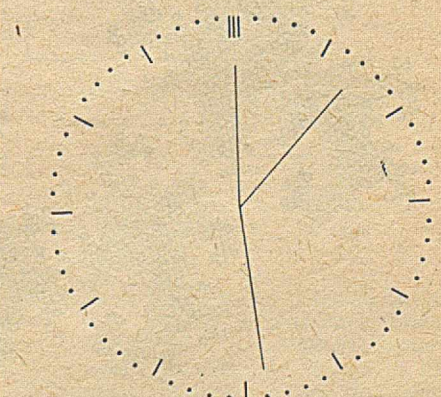
Rys. 4. Krzywe Sierpińskiego



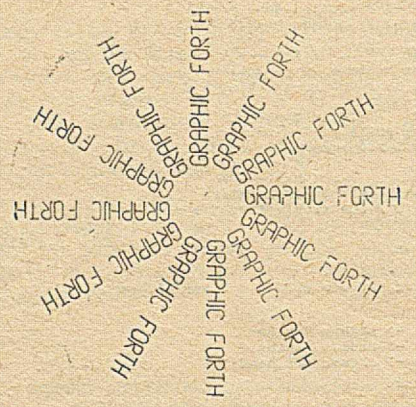
Rys. 5. Płatek śniegu



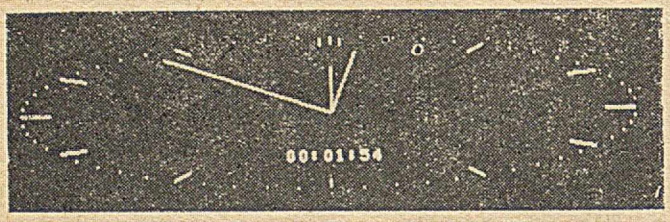
Rys. 6. Zegar wykreślony przez ploter



(trzy ruchome wektory). Program wykreślający zegar przedstawiono na wydruku 5. Składający się na jego definicję program słowa {dial} rysuje tarczę zegarową, program słowa {mark} — zaznacza godzinę dwunastą, słowa {hands} — rysuje wskazówki, zaś słowa {timer} — porusza tymi wskazówkami. Funkcje sumowania spełniają słowa {sum_up} i {sum_down}, niedostępne niestety dla plotera. Wygląd zegara wykreślonego w ten sposób na ekranie monitora graficznego przedstawia fot. 2. Zegar ten może być również wykreślony przez ploter, przy użyciu sekwencji {r dial mark hands} (rys. 6). Zegar z fot. 2 został dodatkowo rozciągnięty, co wymaga zastosowania odpowiedniej transformacji (skalowania).



Rys. 7. Przykład transformacji tekstu



Fot. 2. Rozciągnięty zegar otrzymany na ekranie

Transformacje obrazu

Przedstawiony w pierwszej części artykułu pakiet grafiki żółwia można łatwo rozszerzyć o niektóre transformacje. Obrót praktycznie gwarantują słowa {left} i {right}. Przesunięcie jest niemal identyczne jak ruch naprzód {forward}, musi się tylko wykonywać z uniesionym pisakiem. Skalowanie (a więc wydłużenie, rozszerzenie i powiększenie) można zrealizować mnożąc otrzymane przy użyciu słów {X'} i {Y'} współrzędne przez odpowiednie współczynniki skali. Lustrzane odbicie otrzymuje się zamieniając obrót w lewo {left} na obrót w prawo {right} i odwrotnie.

Teksty alfanumeryczne

Po dobudowaniu transformacji należałoby rozbudować tak powstały język graficzny o możliwość wyprowadzania tekstów. Nie jest to trudne, lecz niestety dość żmudne. Trud się jednak opłaci. Otrzymane bowiem w ten sposób teksty można dowolnie transformować na identycznych zasadach jak pozostałe obiekty graficzne (rys. 7). Program wykreślający taką rozetkę z napisów przedstawiono na wydruku 6.

```
: string_flower
  12 0 do
       home i 30 * right
       text    GRAPHIC FORTH^Z ( delimiter is CTRL-Z )
     loop ;
```

Wydruk 6. Wykreślanie tekstów pod różnymi kątami

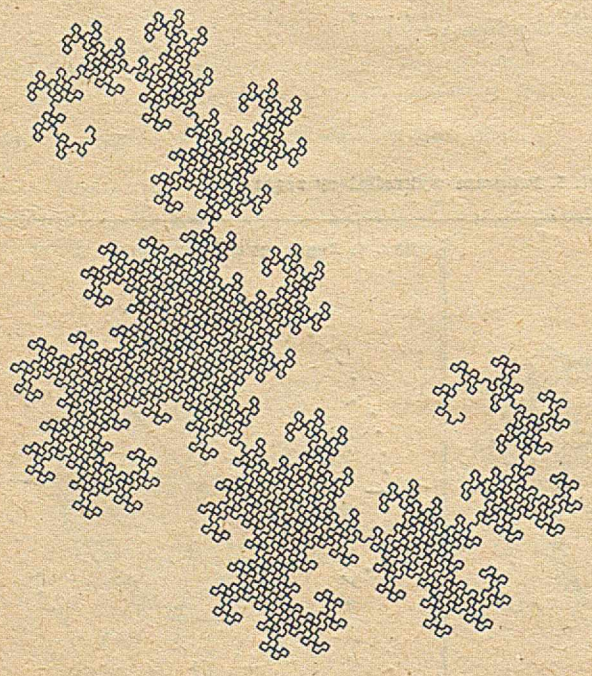
* * *

Przedstawiony w tym artykule pakiet grafiki stanowi podstawę, na której powstał i w dalszym ciągu jest rozwijany język Graphic Forth.

LITERATURA

[1] Szkaradnik Z.: Grafika żółwia w języku Forth. Informatyka, nr 6, 1986.

Jeszcze jeden FRAKTAL



I tym razem, podobnie jak poprzednio, pomysł został zaczerpnięty ze znakomitej pracy B. B. Mandelbrota „The Fractal Geometry of Nature". Przedstawiony tu fraktal, nazwany **dragon**, został narysowany przy użyciu plotera Yokogawa, sterowanego przez mikrokomputer Mera 60. Napisany w języku Graphic Forth program przedstawiono poniżej.

```
: dragon          ( sign level_of_recursion --- )
  dup 0= if
         drop drop 10 forward
       else
         over 45 * turn
         1 over 1- recursive dragon
         over -90 * turn
         -1 over 1- recursive dragon
         drop 45 * turn
       endif ;

1 12 dragon  ( kresli prezentowany fraktal )
```

ZBIGNIEW SZKARADNIK

TADEUSZ MORZY
Instytut Automatyki
Politechnika Poznańska

Synchronizacja transakcji w systemach rozproszonych baz danych (2)

Znane w literaturze algorytmy synchronizacji są realizacjami jednej lub kilku spośród następujących trzech podstawowych metod synchronizacji:

- blokowania,
- porządkowania transakcji według etykiet czasowych,
- walidacji.

Istnieje wiele algorytmów, które stanowią próbę połączenia wymienionych, podstawowych metod synchronizacji. Tworzą one grupę tzw. algorytmów mieszanych. Możliwość tworzenia algorytmów wykorzystujących więcej niż jedną metodę synchronizacji wynika z faktu, że są one w pewnej mierze komplementarne. Poniżej scharakteryzowano trzy podstawowe metody synchronizacji. Bardziej szczegółowy opis tych metod można znaleźć w pracy przeglądowej [1].

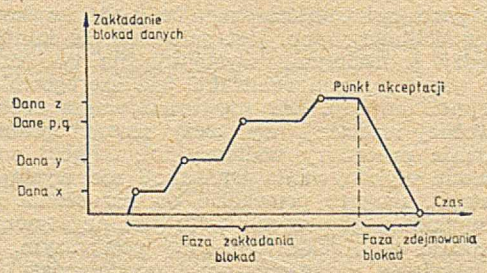
METODA BLOKOWANIA

Metoda blokowania jest najpowszechniej stosowaną metodą synchronizacji. Przyjmuje się w niej, że:

- każda transakcja przed odczytem lub zapisem danej musi założyć na tę daną odpowiednią blokadę,
- blokady wszystkich danych, do których transakcja żądała dostępu, zostaną zdjęte przed zakończeniem jej wykonania.

Zakładanie blokad danych, do których transakcja żąda dostępu, ma na celu niedopuszczenie do odczytu lub zapisu tych danych przez inne transakcje w takich momentach, w których jest możliwe przejściowe naruszenie ograniczeń integralnościowych RBD. Najszerzej stosowanym w praktyce algorytmem opartym na metodzie blokowania jest algorytm blokowania dwufazowego (2PL). Istotą tego algorytmu jest to, aby wykonywanie każdej transakcji przebiegało w dwóch fazach:

- blokowania, kiedy transakcja żąda dostępu do kolejnych danych i nie uwalnia żadnych zasobów,
- odblokowania, kiedy transakcja odblokowuje kolejne dane.

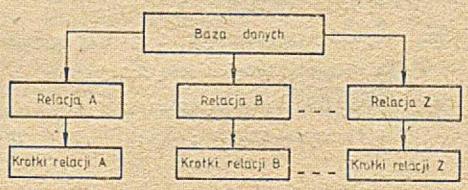


Rys. 1. Schemat realizacji transakcji według algorytmu 2PL

Funkcjonowanie algorytmu 2PL ilustruje rys. 1. Zasadniczą jego zaletą jest to, że może być stosowany dla dowolnych rodzajów zasobów, dowolnych struktur danych i transakcji. Udowodniono, że jest on optymalnym algorytmem blokowania, w sytuacji gdy przetwarzane transakcje mają prawo żądać dostępu do dowolnych danych w systemie, a dostępna informacja o tych transakcjach jest wyłącznie syntaktyczna. Cechą charakterystyczną algorytmu 2PL jest to, że kolejność realizacji zbioru transakcji wynika z przebiegu tej realizacji. Natomiast jego główną wadą jest to, że nie rozwiązuje on samodzielnie problemu konfliktów działania i musi być uzupełniony odpowiednimi metodami ich rozwiązywania. Należy zauważyć, że operacje zakładania i zdejmowania blokad mają właściwości szeregowania par transakcji, a ściślej — szeregują dowolne dwie transakcje, jeżeli istnieje chociaż jedna dana, do której obie żądają dostępu (nawet jeżeli nie prowadzi to do naruszenia kryterium uszeregowalności). Szeregowanie transakcji zmniejsza stopień współbieżności ich wykonywania. Warto wspomnieć, że jeżeli projektant SRBD dysponuje a priori pewną informacją o logicznej strukturze danych (np. zna ograniczenia integralnościowe) lub o fizycznej strukturze danych (np. wie, że dane mają strukturę drzewa), to wymaganie dwufazowej realizacji transakcji może być osłabione.

W literaturze opisano także wiele niedwufazowych algorytmów blokowania. Zasadniczo dotyczą one jednak systemów scentralizowanych baz danych.

Omawiając metody blokowania warto jeszcze wspomnieć o problemie **ziarnistości blokad** i hierarchicznej wersji algorytmu blokowania. Problem ziarnistości blokad jest problemem doboru rozmiaru blokowanej jednostki. Wiąże się z tym, że w SRBD są realizowane transakcje o różnych rozmiarach (rozmiar transakcji jest mierzony rozmiarem zbioru danych, do których transakcja realizuje dostęp). Maksymalizacja współbieżności dla transakcji o małych rozmiarach wymaga małych jednostek blokady, natomiast minimalizacja kosztu blokowania danych dla dużych transakcji wymaga dużych jednostek blokady. Ogólnie akceptowanym rozwiązaniem tego problemu jest przyjęcie koncepcji blokowania hierarchicznego [2]. Zgodnie z tą koncepcją baza danych jest widziana jako hierarchia jednostek blokowania, tak zwanych **ziaren.** Hierarchię ziaren można przedstawić w postaci acyklicznego grafu skierowanego, którego wierzchołki odpowiadają ziarnom, natomiast łuki określają relację zawierania się jednych ziaren w drugich. Zakładając blokadę na określonym poziomie hierarchii blokujemy implicite wszystkie następniki tej jednostki w hierarchii. Ponieważ transakcje realizują blokowanie na różnych poziomach hierarchii blokad, wprowadzono koncepcję **blokad intencjonalnych** [3]. Założenie blokady intencjonalnej danego wierzchołka oznacza, że blokowanie właściwe jest realizowane na niższym poziomie hierarchii blokad.



Rys. 2. Przykładowa hierarchia jednostek blokad

Dla zilustrowania przedstawionych pojęć rozważmy następującą sytuację. Dana jest hierarchia jednostek blokad przedstawiona na rys. 2. Zauważmy, że blokując relację blokujemy jednocześnie wszystkie krotki tej relacji, a blokując bazę danych blokujemy implicite wszystkie relacje należące do tej bazy danych. Jeżeli zostanie zastosowany wyłącznie mechanizm blokad podstawowych (tj. blokada

do odczytu i blokada do zapisu), to przy założeniu nienaruszalności warunku kompatybilności rodzajów blokad, nie można zagwarantować, że w wypadku współbieżnej realizacji transakcji, spójność bazy danych nie zostanie naruszona.

Przykładowo, niech transakcja T_i blokuje pewien zbiór krotek relacji **A** dla zapisu. Załóżmy, że transakcja T_j żąda również dostępu do pewnego zbioru krotek relacji **A** w celu zapisu; ze względu na niekompatybilność blokad żądanie transakcji T_j zostanie odrzucone. Jednakże transakcja T_j może założyć blokadę całej relacji **A** i tym sposobem zablokować również jej krotki. W konsekwencji może to prowadzić do naruszenia ograniczeń integralnościowych nałożonych na bazę danych. Aby uniknąć podobnych sytuacji, wprowadzono wspomniane blokady intencjonalne. Szczegółową prezentację hierarchicznego algorytmu blokowania wykorzystującego koncepcję blokad intencjonalnych można znaleźć w pracy [2].

METODA PORZĄDKOWANIA TRANSAKCJI WEDŁUG ETYKIET CZASOWYCH

Metoda porządkowania transakcji według etykiet czasowych, nazywana w skrócie metodą T/O, stanowi drugą z podstawowych metod synchronizacji transakcji w SRBD. Głównym pojęciem występującym w tej metodzie jest pojęcie etykiety czasowej inicjowania transakcji. Etykieta czasowa inicjowania transakcji T_i, oznaczona przez $TS(T_i)$, jest konkatenacją jednoznacznego identyfikatora stanowiska komputerowego, na którym transakcja T_i jest inicjowana, oraz stanu zegara fizycznego na tym stanowisku w chwili inicjowania transakcji.

Istota metody T/O polega na przyjęciu a priori kolejności realizacji zbioru transakcji w SRBD. Kolejność jest jednoznacznie określona na podstawie etykiet czasowych inicjowania. Zadaniem procedur synchronizacji jest testowanie, czy kolejność nie została w trakcie współbieżnej realizacji zbioru transakcji naruszona (np. na skutek opóźnień w sieci komputerowej).

Podstawową wersję algorytmu realizującego metodę T/O można przedstawić następująco. Z każdą daną **x** należącą do RBD są związane dwie etykiety czasowe: etykieta czasowa odczytu **R_ts(x)** oraz etykieta czasowa zapisu **W_ts(x)**. Przyjmują one wartość etykiety czasowej inicjowania tej transakcji, która jako ostatnia zrealizowała — odpowiednio — odczyt lub zapis tej danej. Procedury odczytu i zapisu dla uproszczonej podstawowej wersji algorytmu opartego na metodzie T/O są przedstawione na rys. 3 i 4.

```
procedure readreq (Ti,x) ;
    if TS(Ti) < W_ts(x)
        then <wycofaj transakcję Ti i wykonaj ją
              z nową etykietą czasową inicjowania> ;
        else <zrealizuj żądanie odczytu > ;
             R_ts(x) : = max {TS(Ti), R_ts(x)} ;
end.
```

Rys. 3. Procedura odczytu dla metody T/O

```
procedure writereq (Ti,x) ;
    if TS(Ti) < R_ts(x) or TS(Ti) < W_ts(x)
        then <wycofaj transakcję Ti i wykonaj ją ponownie
              z nową etykietą czasową inicjowania> ;
        else <zrealizuj żądanie zapisu> ;
             W_ts(x) : = TS(Ti) ;
end.
```

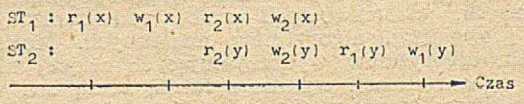
Rys. 4. Procedura zapisu dla metody T/O

W metodzie porządkowania transakcji według etykiet czasowych przyjęty a priori porządek realizacji transakcji prowadzi do wycofywania i restartowania transakcji żądających dostępu do danych niezgodnie z porządkiem etykiet czasowych inicjowania. Sytuacja taka występuje szczególnie często w wypadku transakcji generujących dynamicznie żądania dostępu do danych w trakcie swego wykonywania. Stąd podstawowym problemem metody T/O jest problem stałego restartowania transakcji oraz minimalizacji liczby restartów. Aby zmniejszyć liczbę restartów transakcji, opracowano tzw. konserwatywną wersję metody T/O, w której żądanie dostępu do dowolnej lokalnej bazy danych jest zrealizowane jedynie wówczas, gdy istnieje pewność, że akceptacja tego żądania nie spowoduje restartu jakiejkolwiek transakcji w systemie. Wymaga to komunikacji pomiędzy stanowiskiem komputerowym, na którym realizowane jest żądanie dostępu, a wszystkimi pozostałymi. Podobnie jak w wypadku metody blokowania, w metodzie T/O również występuje problem efektywnej realizacji transakcji o różnych rozmiarach. Jednakże w odniesieniu do metody T/O powyższy problem jest trudniejszy, a znane rozwiązania nie są satysfakcjonujące.

Działanie podstawowego algorytmu metody T/O ilustruje poniższy przykład.

Przykład 1

Rozważmy współbieżną realizację dwóch transakcji T_1 i T_2, które realizują odczyt i zapis danych **x** i **y** zlokalizowanych odpowiednio na stanowisku ST_1 i ST_j. Załóżmy, że etykiety czasowe inicjowania obu transakcji są takie, że $TS(T_1)<TS(T_2)$. Załóżmy, że kolejność zgłaszania żądań odczytu i zapisu przez obie transakcje jest taka, jak przedstawiono na rys. 5.



Rys. 5. Kolejność zgłaszania żądań dostępu do danych przez transakcje z przykładu 1

Symbole r_1, w_1 oraz r_2, w_2 oznaczają zgłoszenie żądania dostępu do danej (r — odczyt, w — zapis) — odpowiednio — przez transakcję T_1 i T_2. Zgodnie z procedurami odczytu i zapisu przedstawionym na rys. 3 i 4, wszystkie żądania dostępu do danej x na stanowisku ST_1 zostaną zaakceptowane. Na stanowisku ST_2 występuje niezgodność kolejności zgłoszeń żądań dostępu z przyjętą a priori kolejnością realizacji. Zgodnie z procedurami odczytu i zapisu żądania $r_1(y)$ i $w_1(y)$ zostaną odrzucone. Spowoduje to konieczność wycofania częściowo zrealizowanej transakcji T_1 i jej restart.

Z przedstawionego przykładu widać, że metoda T/O jest wolna od konfliktów martwego punktu, stałego zablokowania i cyklicznego restartu.

METODA WALIDACJI

Podejścia walidacyjne wyróżniają następujące trzy zasadnicze cechy:

(1) podstawowym sposobem rozwiązywania konfliktów dostępu jest wycofanie transakcji,

(2) decyzja o akceptacji bądź wycofaniu transakcji jest podejmowana po jej wykonaniu,

(3) zapytania do RBD nie są nigdy odrzucane.

Główną przesłanką, która legła u podstaw konstrukcji algorytmów walidacyjnych, jest założenie, że konflikty dostępu między transakcjami występują na tyle rzadko, że większość współbieżnych realizacji jest szeregowalna i nie wymaga synchronizacji. Dla zagwarantowania uszeregowalności dowolnej realizacji wystarczy zatem ograniczyć się do wykonania procedury walidacyjnej, dającej odpowiedź, czy w trakcie wykonywania transakcji nie wystąpiły konflikty dostępu, mogące prowadzić do naruszenia spójności RBD. W wypadku, gdy wynik testu jest negatywny, transakcję wycofuje się i restartuje. Przy powyższym, optymistycznym założeniu odnośnie prawdopodobieństwa wystąpienia konfliktów dostępu, liczba restartów powinna być niewielka.

Istnieje wiele możliwych sposobów (algorytmów) realizacji metody walidacji. Historycznie pierwsze algorytmy walidacyjne oparte były na konstrukcji grafu relacji określającej wzajemną kolejność realizacji żądań dostępu współbieżnie realizowanych transakcji. Acykliczność grafu tej relacji jest, jak wiadomo, warunkiem dostatecznym uszeregowalności zbioru współbieżnie realizowanych transakcji. Uaktualnienie tego grafu następuje każdorazowo w wypadku realizacji żądania odczytu lub zapisu danej w RBD.

Przykładowo, graf relacji poprzedza dla przedstawionego wyżej przykładu ma postać:

$$T_1 \xrightarrow[x]{} T_2 \xrightarrow[y]{} T_1.$$

Nie jest on acykliczny, stąd wynika konieczność wycofania jednej z transakcji. Podstawowym problemem związanym z implementacją tego rodzaju algorytmów w SRBD jest problem efektywnej konstrukcji i uaktualniania grafu relacji poprzedzania.

Nowsze algorytmy walidacyjne są oparte na koncepcji wprowadzenia dodatkowej fazy realizacji transakcji, właśnie fazy walidacji. Dobrą ilustrację tej idei stanowi algorytm Kunga-Robinsona. W algorytmie tym wykonywanie transakcji przebiega w trzech fazach: odczytu, walidacji oraz zapisu. W fazie odczytu transakcja jest wykonywana lokalnie, w obszarze roboczym lokalnej bazy danych. W momencie zakończenia wykonywania transakcji (zakończenie to jest rozumiane jako zakończenie wykonywania transakcji logicznej) następuje przejście do fazy drugiej — fazy walidacji. W tej fazie sprawdza się, czy zaakceptowanie transakcji nie naruszy spójności RBD, to znaczy, czy po zaakceptowaniu transakcji wynikowa realizacja (uwzględniająca działanie akceptowanej transakcji) pozostanie szeregowalna. Jeżeli realizacja pozostanie szeregowalna, to transakcja jest akceptowana i przechodzi do wykonywania fazy zapisu. W przeciwnym razie następuje wycofanie transakcji i jej restart.

Podstawowe zalety metody walidacji są następujące:

(i) brak opóźnienia w wykonywaniu tych transakcji, które nie naruszają spójności RBD (dotyczy to w szczególności zapytań),

(ii) brak występowania konfliktów martwego punktu, cyklicznego restartu i stałego zablokowania.

Jednakże alogrytm Kunga-Robinsona (podobnie jak inne algorytmy metody walidacji) w celu walidacji transakcji wymaga dysponowania pełną i aktualną informacją o globalnym stanie całego systemu. Informacja ta może być gromadzona na jednym wybranym stanowisku komputerowym SRBD lub na wszystkich stanowiskach. Można zatem stwierdzić, że efektywne rozwiązanie problemu walidacji, nie wymagające każdorazowo kosztownej komunikacji między rozproszonymi stanowiskami SRBD, warunkuje przydatność metody walidacji w SRBD. Wniosek ten potwierdzają badania symulacyjne i analityczne dotyczące oceny efektywności metod synchronizacji w SRBD.

Metoda walidacji gwarantuje też wysoką efektywność w odniesieniu do systemów scentralizowanych.

AKTUALNE TENDENCJE W ZAKRESIE ALGORYTMÓW SYNCHRONIZACJI W SRBD

Aktualnie prowadzone prace dotyczące problematyki synchronizacji w systemach baz danych, zarówno scentralizowanych, jak i rozproszonych, koncentrują się zasadniczo na trzech kierunkach badawczych:

(1) konstruowaniu nowych algorytmów, stanowiących integrację podstawowych metod synchronizacji, z bezpośrednim włączeniem do nich aspektu niezawodnościowego,

(2) konstruowaniu algorytmów synchronizacji z uwzględnieniem semantyki operacji na danych,

(3) konstruowaniu algorytmów synchronizacji dla tzw. wielowersyjnych systemów baz danych.

Integracja metod synchronizacji

Dążenie do zintegrowania podstawowych metod synchronizacji i opracowania metod hybrydowych wynika z faktu, iż wady i zalety tych metod są w dużym stopniu komplementarne. Szacuje się, że ogólna liczba hybrydowych algorytmów synchronizacji przekracza tysiąc. Algorytmy hybrydowe zapewniają wyższą efektywność działania SRBD i rozwiązują najczęściej wszystkie konflikty działania SRBD. Łączne rozpatrywanie problemów spójności i konfliktów działania SRBD stanowi jedną z zasadniczych zalet algorytmów hybrydowych. W ostatnim czasie szczególnego znaczenia nabiera aspekt niezawodności algorytmów synchronizacji w SRBD. W zasadzie dotyczy to częściowo lub całkowicie duplikowanych RBD. Jako ilustrację problemu rozważmy prosty przykład.

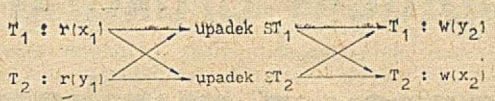
Przykład 2

Załóżmy przypadek duplikowanej RBD, w której zarządzanie współbieżnym dostępem do danych jest realizowane zgodnie z algorytmem 2PL. Zarządzanie kopiami fizycznymi tej samej danej logicznej realizuje się następująco. W celu odczytu danej logicznej **X** realizowany jest odczyt dowolnej dostępnej kopii fizycznej x tej danej, natomiast w celu uaktualnienia stanu danej logicznej **X** należy uaktualnić stan wszystkich dostępnych kopii fizycznych tej danej. Jeżeli kopia danej nie może zostać uaktualniona na skutek awarii stanowiska, to jest ona ignorowana. W momencie usunięcia awarii stan tej kopii jest uaktualniany odpowiednio do bieżącego stanu pozostałych kopii tej danej (w celu zapewnienia spójności zewnętrznej).

Można łatwo wykazać, że przedstawiony algorytm aktualizacji kopii może prowadzić, w razie wystąpienia awarii stanowisk, do naruszenia spójności RBD. Rozważmy SRBD zawierającą dwie dane logiczne **X** i **Y** posiadające kopie fizyczne x_1, x_2, y_1, y_2, przechowywane na stanowiskach, odpowiednio: ST_1, ST_2, ST_3 i ST_4. Rozpatrzmy współbieżną realizację dwóch transakcji T_1 i T_2, takich że:

T_1 realizuje odczyt danej **X** i następnie zapis **Y**,

T_2 realizuje odczyt danej **Y** i zapis danej **X**. Załóżmy, że w celu realizacji odczytu transakcji T_1 i T_2 SZRBD wybrał kopie fizyczne x_1 i y_1. Załóżmy następnie, że po zrealizowaniu odczytu danych x_1 i y_1 przez transakcje T_1 i T_2 na stanowiskach ST_1 i ST_3, nastąpił upadek tych stanowisk. Transakcje T_1 i T_2 realizują zapis danych logicznych **Y** i **X**, a ściślej, zapis jedynych dostępnych kopii y_2 i x_2. Obie transakcje uzyskają założenie blokad do zapisu tych danych. Przebieg przedstawionej realizacji zilustrowano na rys. 6. Strzałki na rysunku reprezentują relację poprzedzania, tj. kolejność zachodzenia zdarzeń w czasie. Przedstawiona realizacja jest niepoprawna w tym sensie, że nie spełnia kryterium uszeregowalności. Obie transakcje znajdują się bowiem w konflikcie i wymagają synchronizacji dostępu do danych.



Rys. 6. Realizacja transakcji z przykładu 2

Z przykładu 2 widać wyraźnie, że istnieje potrzeba bezpośredniego włączenia mechanizmów niezawodnościowych do mechanizmów synchronizacyjnych.

Wykorzystanie wiedzy o danych i transakcjach

Inną tendencją w aktualnie prowadzonych pracach badawczych jest włączenie do procesu synchronizacji wiedzy użytkownika o danych i o przetwarzanych transakcjach, określane skrótowo mianem włączania semantyki. Zasadnicze znaczenie dla rozwoju tego kierunku badawczego miały prace prowadzone w zakresie konstruowania oprogramowania dla systemów rozproszonych, w szczególności prace poświęcone pojęciu transakcji. Wskazały one potrzebę modyfikacji trzech aspektów dotyczących przyjmowanego modelu transakcji.

Pierwsza modyfikacja polega na rozszerzeniu zbioru operacji składających się na transakcje. Ma to na celu definiowanie transakcji w języku operacji specyficznych dla obiektów, na których transakcje operują, a więc za pomocą pojęć takich, jak: kolejki, katalogi, dane, procedury, bufory, itp., a nie za pomocą prostych operacji odczytu i zapisu. Inaczej mówiąc, transakcja operująca danym obiektem (np. kolejką) wykorzystuje operacje manipulacji specyficzne dla tego obiektu (np. utwórz kolejkę, pobierz z kolejki, itp.).

Druga modyfikacja polega na takim rozszerzeniu modelu transakcji, aby umożliwić wzajemną komunikację i kooperację transakcji. Pozwala to odejść od dotychczasowej koncepcji, zgodnie z którą zadanie użytkowe jest realizowane przez pojedynczą transakcję lub zbiór niezależnych transakcji. Jest to istotne, ponieważ istnieje wiele zadań, których specyfika wskazuje na celowość ich realizacji właśnie przez zbiór wzajemnie zależnych, kooperujących transakcji, a nie przez transakcje niezależne.

Trzecia modyfikacja modelu transakcji polega na dopuszczeniu, w ciele transakcji, inicjowania kolejnych transakcji

— tzw. transakcji zagnieżdżonych. Włączenie semantyki do procesu synchronizacji pozwala jednocześnie zwiększyć zarówno stopień współbieżności realizacji transakcji, jak i stopień odporności algorytmu na awarie systemu. Ten kierunek badań jest stosunkowo nowy i można mówić wyłącznie o wstępnych wynikach prac w tym zakresie. Jednak wydaje się, że jest bardzo perspektywiczny i prawdopodobnie w najbliższym czasie prowadzone badania przyniosą znaczące wyniki.

Algorytmy wielowersyjne

Na zakończenie należy jeszcze wspomnieć o bardzo atrakcyjnych, zarówno z teoretycznego, jak i praktycznego punktu widzenia, kierunku badań w zakresie synchronizacji, a mianowicie o algorytmach wielowersyjnych. Idea wielowersyjnych algorytmów synchronizacji jest następująca. Transakcja żądająca dostępu do danej x w celu zapisu, nie uaktualnia stanu danej x, lecz tworzy jej nową wersję. Dlatego kolejna transakcja żądająca dostępu do danej x w celu jej odczytu ma możliwość odczytu jednej z dwóch aktualnie dostępnych wersji tej danej.

Atrakcyjność algorytmów wielowersyjnych wynika z trzech przesłanek:

(1) pozwalają na znaczne zwiększenie stopnia współbieżności realizowanych transakcji,

(2) pozwalają w naturalny sposób połączyć mechanizmy synchronizacji z mechanizmami niezawodności (nie zachodzi potrzeba prowadzenia dzienników uaktualnień),

(3) można w nich łatwo zagwarantować, że zapytania do RBD nie będą nigdy opóźniane ani odrzucane.

* * *

Czytelników zainteresowanych bardziej szczegółową i pogłębioną prezentacją problematyki synchronizacji transakcji w SRBD odsyłamy do prac [1, 3—5].

LITERATURA

[1] Cellary W., Morzy T.: Problemy i metody synchronizacji w systemach rozproszonych baz danych. Archiwum Automatyki i Telemechaniki, t. XXX, nr 5—6, 1986.

[2] Gray J.: Notes on data base operating systems. Pp. 393—481, Bayer R., Graham R. M., Seegmuller G. (eds.), Operating Systems — An Advanced Course. Springer-Verlag, Berlin, 1978.

[3] Ceri S., Pelagatti G.: Distributed Databases — Principles and Systems. McGraw-Hill, New York, 1984.

[4] Bernstein P. A., Goodman N.: Concurrency Control in Distributed Database Systems. ACM Computing Surveys, Vol. 11, No. 2, pp. 185—221, 1981.

[5] Bernstein P. A., Goodman N.: Multiversion Concurrency Control — Theory and Algorithms, ACM Trans, on Database Systems, Vol. 8, No. 4, pp. 465—483, 1983.

CZESŁAW RATKA
Gliwice

Adaptacja drukarki wierszowej DW 204-2 do pracy z minikomputerem SM-4

Możliwość wykorzystania urządzeń zewnętrznych zbudowanych według jednego standardu do pracy w innym standardzie stanowi problem, z którym zetknęło się wielu użytkowników. Różnice, choćby drobne, w realizacji technicznej różnych standardów uniemożliwiają bezpośrednie dołączenie urządzenia zewnętrznego opartego na innym standardzie. Sam układ sprzęgający danego urządzenia stanowi jednak tylko jego fragment i koszt budowy lub przebudowy tego układu jest nieporównywalnie niski z ceną całego urządzenia. Często przy niewielkich nakładach można dokonać adaptacji urządzenia przeznaczonego do pracy w jednym standardzie do pracy w innym. Chodzi tu zwłaszcza o urządzenia o stosunkowo prostym protokole transmisji, jak drukarki, czytniki taśmy papierowej, perforatory itp. W wielu wypadkach warto również zastanowić się nad dołączeniem tego rodzaju urządzeń jednocześnie do dwóch zestawów komputerowych.

ZASADY OGÓLNE

Różnice budowy urządzeń zewnętrznych przeznaczonych do pracy w różnych standardach mogą być czasami znaczne i już na podstawie wstępnego zapoznania się z nimi można określić opłacalność przystosowywania danego urządzenia do wymagań określonego standardu. Często różnice te są niewielkie. Niekiedy problem prowadzenia transmisji można rozwiązać za pomocą oprogramowania, na przykład wówczas, gdy konieczna jest jedynie zmiana kodu transmitowanych znaków. Czasami jest potrzebne tylko zanegowanie niektórych sygnałów; innym razem konieczna jest zmiana przebiegów czasowych sygnałów sterujących transmisją. Może też zachodzić potrzeba zmiany poziomu napięciowego sygnałów. Każdy przypadek adaptacji należy rozpatrywać indywidualnie.

Adaptacja urządzenia do danego zestawu komputerowego to zastąpienie w tym zestawie urządzenia, które mogłoby i powinno się w nim znajdować, urządzeniem, którego konstrukcja nie przewiduje pracy w danym standardzie. Przystępując do określenia warunków takiej zamiany, należy rozpatrzeć różnice w budowie urządzenia zastępującego i zastępowanego. W przypadku transmisji równoległej linie sprzężenia urządzenia wejścia-wyjścia można podzielić na trzy grupy:

- linie nawiązania transmisji określające gotowość, prawidłowy stan urządzenia, sprawność wszystkich układów;
- linie danych do przesyłania znaków w odpowiednim kodzie;
- linie sterowania transmisją, takie jak żądanie nadawania, potwierdzenie odbioru itp.

Powyższa klasyfikacja jest celowo bardzo ogólna, ze względu na różnorodność spotykanych rozwiązań technicznych i służy tylko do wyraźniejszego przedstawienia zadania.

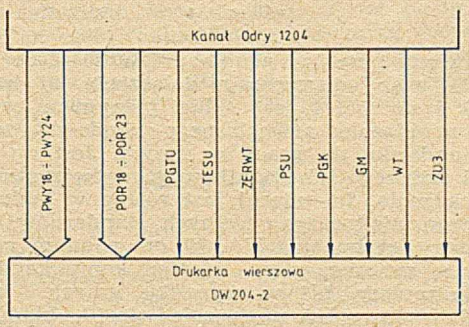
Po rozpatrzeniu różnic części sprzęgających obu urządzeń, należy określić możliwość i sposób ich eliminacji. W przypadku różnic w kodach przesyłanych znaków trzeba zbudować odpowiedni transkoder. Różnice w pozostałych sygnałach trzeba analizować w powiązaniu ze sposobem prowadzenia transmisji w obu urządzeniach. Należy brać pod uwagę zarówno przebiegi czasowe, jak i protokoły przesyłania. Problem dopasowania sprowadza się zawsze do takiego przekształcenia odpowiednich sygnałów, aby urządzenie wejścia-wyjścia z jednej, a komputer z drugiej strony, rozpoznawały w nich własne sygnały. Zadanie to spełnia układ nazywany dalej układem adaptującym (UA).

OPIS UKŁADU ADAPTUJĄCEGO

Poniżej przedstawiono opis UA umożliwiającego pracę drukarki wierszowej DW 204-2, przeznaczonej do współpracy z Odrą 1204, w zestawie minikomputera SM-4. Różnice konstrukcyjne i organizacyjne obu tych komputerów są

znaczne i szczegółowe ich omówienie wykracza poza ramy tego artykułu. W skrócie przedstawiają się one następująco:

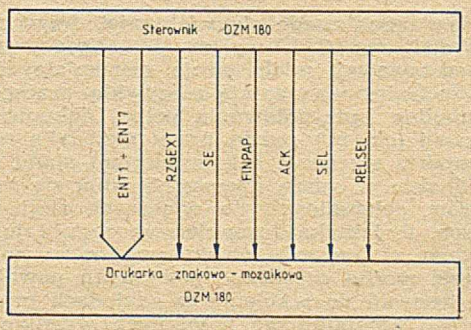
1. Sygnały drukarki DW 204-2, przedstawione na rysunku 1, są sygnałami kanału Odry 1204; inaczej mówiąc, jednostka centralna porozumiewa się bezpośrednio z urządzeniem wejścia-wyjścia. Natomiast w wypadku minikomputera SM-4 każde urządzenie zewnętrzne ma swój sterownik, który nawiązuje i prowadzi transmisję z jednostką centralną przez wspólną szynę typu Unibus (WS). Sygnały urządzenia nie są identyczne z sygnałami WS. Wygląda to tak, jakby urządzenie wejścia-wyjścia Odry 1204 zawierało w sobie odpowiednik, choć organizacyjnie i architektonicznie dość odległy, SM-owskiego sterownika urządzenia zewnętrznego.



Rys. 1. Linie sprzęgające drukarki wierszowej DW 204-2

2. Drukarka DW 204-2 ma repertuar 64 znaków (6 linii danych) o kodach różnych od kodów ASCII stosowanych w minikomputerze SM-4. Brakuje wśród nich znaków sterujących wysuwem papieru — jak LF lub FF, są tylko znaki do wydruku. W związku z tym, organizacja transmisji danych do drukarki jest w obu wypadkach różna, choć przebiega tak samo znak po znaku i kolejne znaki są zapamiętywane w buforze drukarki. Dla drukarki współpracującej z minikomputerem SM-4 sygnałem do rozpoczęcia drukowania jest pojawienie się znaku sterującego, np. CR, LF, FF, zaś DW 204-2 rozpoczyna wydruk zawartości bufora w momencie zaniku sygnału PSU, co informuje o zakończeniu transmisji. W tym wypadku transmisja jest nawiązywana na nowo dla wydruku każdego kolejnego wiersza i za każdym razem przed pojawieniem się pierwszego znaku danych do wydruku, najpierw jest przesyłany specjalny znak sterujący, nie wpisywany do bufora drukarki. Bity tego znaku określają sposób wysuwu papieru. Wysuw rozpoczyna się natychmiast po przyjęciu przez drukarkę znaku sterującego i może odbywać się równolegle z przyjmowaniem przez bufor danych do wydruku.

3. Drukarka DW 204-2 jest zbudowana z układów tranzystorowych. Sygnały logicznego zera i jedynki mają odpowiednio wartość 0 V i —6 V, zaś wszystkie urządzenia minikomputera SM-4 są zbudowane z zastosowaniem układów scalonych TTL.



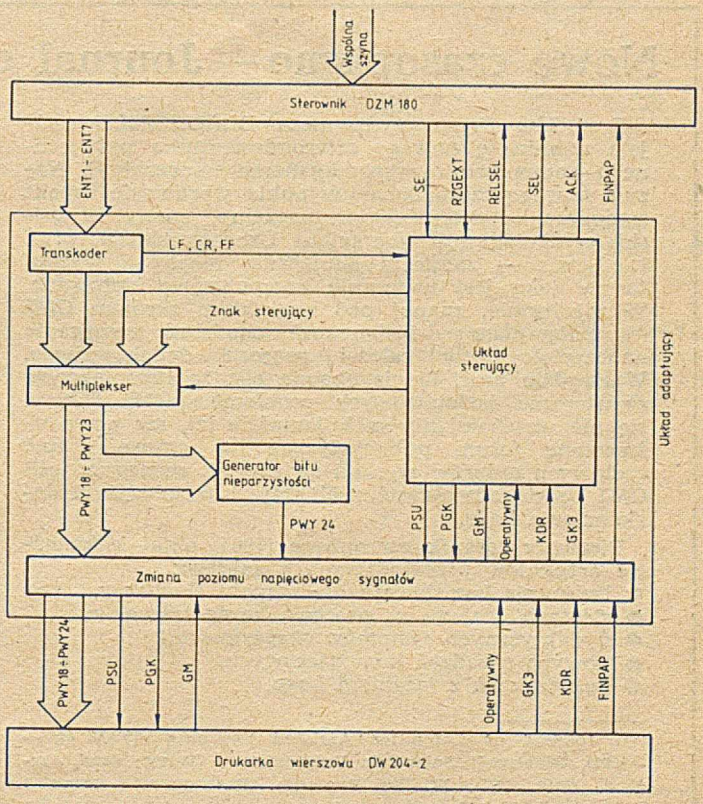
Rys. 2. Linie sprzęgające drukarki znakowo-mozaikowej DZM 180

Chcąc przystosować drukarkę DW 204-2 do współpracy z SM-4 należy zbudować taki UA, który spełniałby zadania sterownika urządzenia zewnętrznego, z jednej strony komunikując się ze wspólną szyną, zaś z drugiej sterując pracą drukarki. Aby uniknąć budowy nowego sterownika, a jednocześnie zminimalizować listę zadań UA, zdecydowano się wykorzystać sterownik drukarki mozaikowej DZM 180, stosowany w zestawach minikomputera SM-4. Na rysunku 2 przedstawiono linię sprzężenia drukarki DZM 180. Uwidoczniono tylko sygnały wykorzystane w realizacji omawianego układu, tj. te, których obecność jest niezbędna do prawidłowej pracy sterownika. Zadania UA polegają na przekształcaniu sygnałów na drodze od sterownika DZM 180 do drukarki DW 204-2. Trzeba więc wśród linii z rysunków 1 i 2 wyszukać pary odpowiedników lub przynajmniej sygnały o podobnych funkcjach; zestawiono je w poniższej tabeli. Przedstawione pary nie stanowią ścisłych odpowiedników, lecz możliwe jest ich dopasowanie. Przykładem są tu linie przesyłania danych. UA należy wyposażyć w odpowiedni transkoder.

Odpowiadające sobie linie drukarek DW 204-2 i DZM 180

Drukarka wierszowa DW 204-2	Drukarka znakowo-mozaikowa DZM 180
PWY18 — PWY24 sześć linii do przesłania znaku danych oraz bit nieparzystości tego znaku	ENT1 — ENT7 siedem linii do przesyłania znaku danych w kodzie ASCII
PSU sygnał rozpoczynający transmisję i informujący o jej trwaniu	RZGEXT sygnał zerowania ogólnego
PGK sygnał strobu dla znaku danych	SE sygnał strobu dla znaku danych
GM sygnał informujący jednostkę centralną o gotowości do przyjęcia znaku danych	ACK potwierdzenie przyjęcia znaku danych

Większa liczba sygnałów drukarki DW 204-2 w porównaniu z drukarką DZM 180 wynika z faktu, że DW 204-2 pracuje bezpośrednio w kanale Odry 1204. Część z nich służy do nawiązania i kontroli transmisji w kanale. Są to sygnały określające numer drukarki w kanale, stan wskaźników drukarki, numer testowanego wskaźnika, rodzaj rozkazu realizowanego przez drukarkę, przerwanie. W wypadku drukarki DZM 180 odpowiedniki funkcji realizowanych przez te sygnały wykonuje sterownik, więc przy dołączeniu DW 204-2 do minikomputera SM-4 nie będą one wykorzystywane, stąd ich brak na rysunku 3, przedsta-

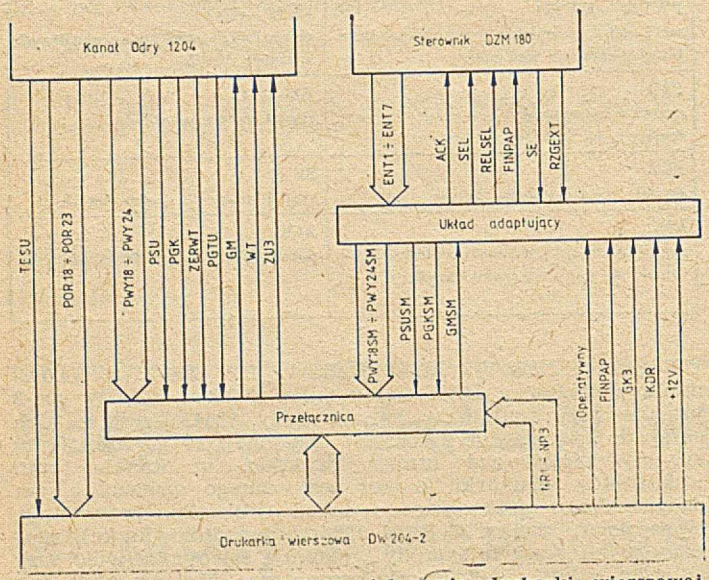


Rys. 3. Schemat dołączenia drukarki wierszowej DW 204-2 do minikomputera SM-4

wiającym schemat takiego połączenia. Występują natomiast dodatkowe sygnały między DW 204-2 a UA, które normalnie, tj. przy pracy DW 204-2 w kanale Odry 1204, nie są jej sygnałami. Sygnały te są potrzebne do właściwego działania Układu Sterującego, którego zadania polegają na realizacji odpowiednich protokołów transmisji w obu kierunkach.

PRZEŁĄCZANIE PRACY

Jak już wspomniano, w niektórych wypadkach warto rozpatrzeć realizację jednoczesnego dołączenia danego urządzenia wejścia-wyjścia do dwóch zestawów komputerowych. Wymaga to zbudowania przełącznicy, która łączy linie urządzenia na przemian z liniami jednego lub drugiego komputera. Sam pomysł takiego połączenia nie jest nowy, lecz jego realizacja wiąże się z problemem adaptacji danego urządzenia do innego standardu. Chodzi bowiem o to, aby wraz z adaptacją do pracy w pewnym zestawie komputerowym nie rezygnować z obecności tego urządzenia w dotychczasowej konfiguracji sprzętowej. Jest to oczywiście celowe w wypadku, gdy oba zestawy komputerowe znajdują się w bezpośrednim sąsiedztwie.



Rys. 4. Schemat jednoczesnego dołączenia drukarki wierszowej DW 204-2 do Odry 1204 i SM-4

W omawianym przykładzie adaptacji drukarki DW 204-2 do zestawu minikomputera SM-4 chodzi o zachowanie kontaktu tej drukarki z Odrą 1204. Schemat jednoczesnego dołączenia drukarki DW 204-2 do obu komputerów przedstawiono na rysunku 4. Sygnały drukarki są przez układ przełącznicy kierowane bądź do kanału Odry 1204, bądź do UA. W celu zaznaczenia odrębności sygnałów PWYn, PSU, PGK, GM, służących do komunikacji z Odrą 1204, od łączących drukarkę z UA, te ostatnie zaopatrzono dodatkowymi literami SM. Przejścia przez układ przełącznicy wymagają tylko te sygnały drukarki, które mogłyby zmienić jej stan lub zmienić stan istniejący w jednym z komputerów wtedy, gdy drukarka jest akurat od niego odłączona.

* * *

Przedstawiony przykład adaptacji DW 204-2 do minikomputera SM-4 oraz jednoczesnego dołączenia tej drukarki do Odry 1204 i do SM-4 jest tylko jednym z przypadków adaptacji urządzenia zewnętrznego zbudowanego według jednego standardu, do pracy w innym. Jest on ilustracją możliwości szerszego lub dodatkowego wykorzystania określonego urządzenia. Im więcej jest różnic w budowie i organizacji transmisji rozpatrywanych standardów, tym bardziej złożona jest budowa UA. W omawianym przykładzie różnice te są znaczne, więc i stopień komplikacji UA jest tu o wiele wyższy niż w wypadku innych adaptacji, a mimo to do budowy UA użyto zaledwie 16 układów TTL o średnim stopniu scalenia. Opłacalność adaptacji drukarki DW 204-2 do minikomputera SM-4 nie ulega więc wątpliwości. Adaptacji takiej dokonano w Ośrodku Badawczo-Rozwojowym Dźwignic i Urządzeń Transportowych DETRANS w Bytomiu.

LITERATURA

[1] Drukarka Wierszowa DW 204-2. Dokumentacja Techniczno-Ruchowa, część I. Wrocławskie Zakłady Elektroniczne ELWRO.

[2] Drukarka Znakowo-Mozaikowa DZM 180. Opis Techniczny. Zakłady Mechaniczno-Precyzyjne Mera-Błonie.

[3] Dokumentacja Techniczno-Ruchowa Systemu Minikomputerowego. Cz. I, tom 7. Dokumentacja modułu drukarki DZM 180. Centrum Naukowo-Produkcyjne Technik Komputerowych i Pomiarów, Warszawa, 1982.

Nowe czasopismo — Journal of Computer Assisted Learning

Brytyjska firma wydawnicza BLACKWELL podjęła publikację nowego czteromiesięcznika poświęconego komputerowo wspomaganemu nauczaniu [1]). Nazwa nowego czasopisma uwypukla wprawdzie aspekt samouczenia, ale redaktor naczelny, znany brytyjski psycholog z Uniwersytetu Lancasterskiego, prof. R. Lewis — zamierza skutecznie lansować wyższą formę, jaką jest nauczanie komputerowo sterowane. W tej formie, znanej pod angielskim skrótem CMI (computer-managed-instruction), chodzi o pomaganie nauczycielowi w całości procesu dydaktycznego. Wprawdzie prof. Lewis obawia się, że kompleksowe rozwiązanie podstawowych problemów CMI dokona się nie wcześniej niż za kilkanaście lat, ale nowo utworzone forum publicystyczne ma szansę skupić czołowych autorów w skali światowej, zwłaszcza jeśli CMI będzie obejmować problemy dokształcania i doskonalenia kadr.

Pierwszy rocznik czasopisma (1985) objął kilkanaście artykułów przeglądowych z zakresu:

- tworzenia baz danych dla zespołu słuchaczy,
- strategii dialogu w procesie korepetytorskim,
- dydaktycznych systemów ekspertowych,
- nowych urządzeń komputerowych,
- doświadczeń z językiem Logo,
- stosunku słuchaczy do nowych form dydaktycznych,
- posługiwania się przez słuchaczy autoarkuszem obliczeniowym,
- nowo publikowanych kaset programowych i książek,
- perspektyw tzw. inteligentnego wspomagania procesu dydaktycznego.

Przesadne ograniczanie się większości dotychczasowych specjalistów CAL wyłącznie do problematyki szkoły podstawowej, miało swoje zrozumiałe źródła. Autorzy zachodni wymieniają niezwykłą pracochłonność i żmudne przygotowania pełnej dokumentacji dydaktycznej (ok. 100 godzin pracy metodyka na godzinę kontaktu słuchacza z komputerem) planu szkolenia. Jednakże w miarę upowszechniania się mikrokomputerów dołączalnych do większych sieci komputerowych, dydaktyka komputerowa została już zaakceptowana w kręgach wojskowych, a w najbliższym czasie można oczekiwać pierwszych poważniejszych zastosowań do szkolenia w zakładach pracy.

W krajach komputerowo zacofanych wszelkie rozważania na temat systemów ekspertowych w dydaktyce wydają się oderwane od rzeczywistości. Jednakże pojawienie się pierwszych komputerowo wspomaganych kursów (z zakresu bhp czy też określonych zbiorów przepisów prawnych) jest tylko kwestią czasu.

ADAM B. EMPACHER

[1]) Journal of Computer Assisted Learning (ISSN 0266—4909), Blackwell Scientific Publications, Osney Mead, Oxford, OX2 OEL, Wielka Brytania. Prenumerata tomu 1 (1985) wyniosła 7,50 funtą szterlinga.

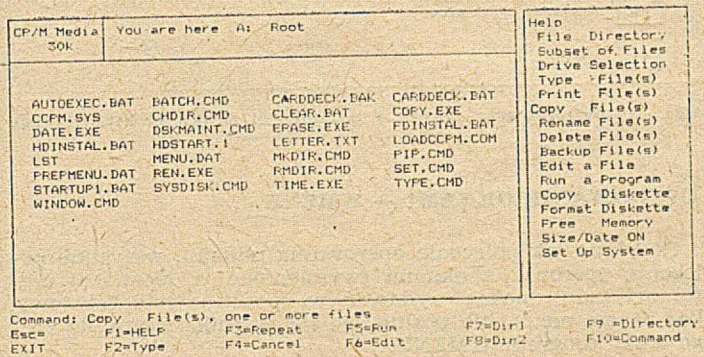
JANUSZ ZALEWSKI
Warszawa

Concurrent DOS (2)

W dokończeniu opisu systemu operacyjnego Concurrent DOS przedstawiono bardziej szczegółowe omówienie ważniejszych poleceń realizowanych techniką menu. Dotyczą one zarządzania plikami, operowania oknami, tworzenia list menu oraz zarządzania drukowaniem i są wykonywane odpowiednio przez programy: FM, WMENU, EDITMENU i PRINTMGR.

ZARZĄDZANIE PLIKAMI — FM

Tradycyjnie, do najczęściej wykonywanych czynności systemu operacyjnego należą operacje dotyczące plików. Ich obsługę zapewnia pojedynczy program zwany zarządcą plików, FM (ang. file manager). Wywołanie programu FM powoduje wyświetlenie na ekranie (podzielonym na tzw. panele, por. cz. 1 artykułu) pełnej zawartości dysku i listy dostępnych poleceń (rys. 1). Oprócz najbardziej typowych poleceń operowania plikami (TYPE, PRINT, COPY, RENAME, DELETE, BACKUP, EDIT, DISKCOPY, FORMAT, grupa poleceń dotyczących skorowidzów) w systemie Concurrent DOS istnieją polecenia nie spotykane u jego poprzedników.

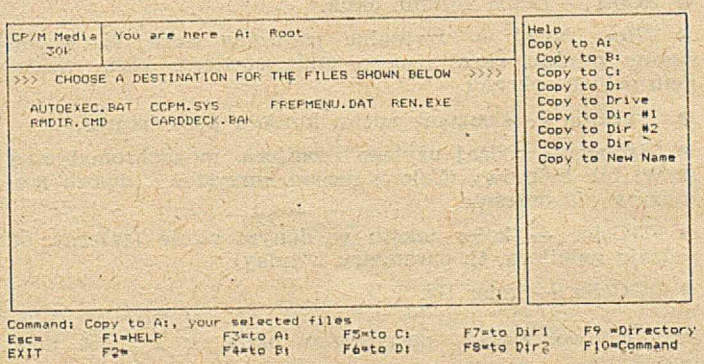


Rys. 1. Menu zarządcy plików — FM

Spośród zalet operowania techniką menu dla tego rodzaju poleceń można wyliczyć dwie:

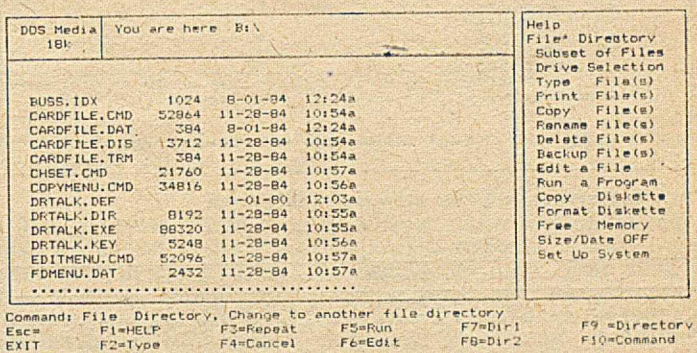
— możliwość operowania plikami bez konieczności pamiętania formatu i parametrów poleceń

— możliwość wyboru grup plików bez użycia zmiennych nazw (ang. wildcards) „*" i „?", lecz za pomocą klawiszy INSERT i DELETE. Te cechy są szczególnie dogodne dla polecenia o wielu możliwościach, jak np. COPY (rys. 2).

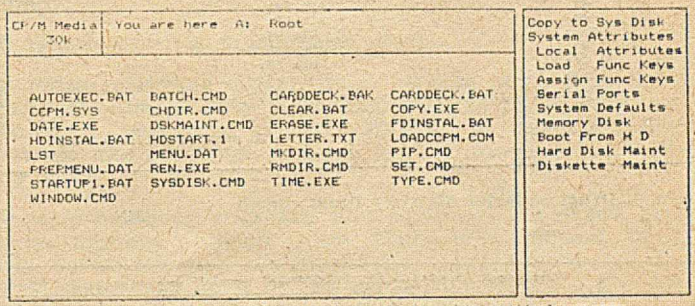


Rys. 2. Wybór plików dla polecenia COPY

Jednym z nowych, choć nie tak istotnych poleceń, jest polecenie uwzględnienia długości i daty pliku, SIZE/DATE. Jego użycie powoduje włączenie lub wyłączenie podawania informacji o długości i dacie utworzenia pliku, istotnej na przykład w wykonywaniu polecenia DIR (rys. 3).



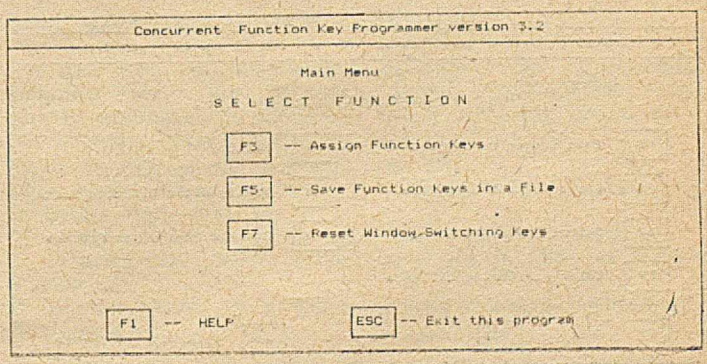
Rys. 3. Wynik wykonania polecenia SIZE/DATE



Rys. 4. Menu polecenia SETUP

O wiele ważniejsze jest polecenie SETUP, służące dokonywaniu zmian parametrów systemowych. Listę menu dla tego polecenia przedstawiono na rys. 4. Obejmuje ona w kolejności m.in.:

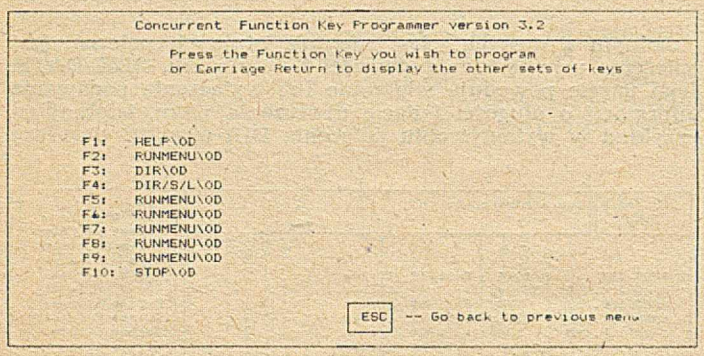
— kopiowanie wybranych plików na dysk systemowy w celu lepszego udostępnienia ich programom we wszystkich oknach (dysk systemowy jest przeszukiwany zawsze, jeśli określonych programów nie ma w bieżącym skorowidzu);



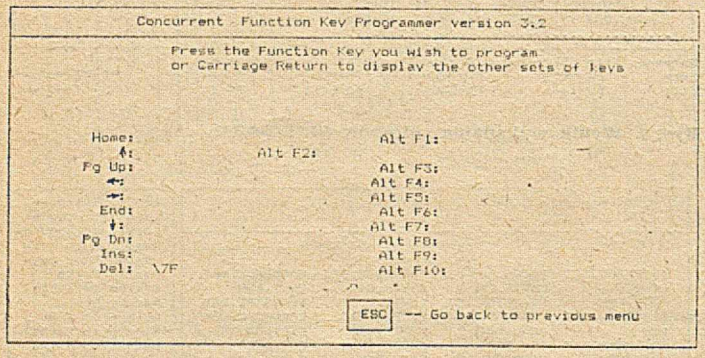
Rys. 5. Definiowanie klawiszy funkcyjnych

— nadawanie plikom domyślnych atrybutów SYSTEM (systemowy) lub LOCAL (lokalny); atrybut SYSTEM umożliwia jednoczesne używanie pliku we wszystkich oknach i musi być zmieniony na LOCAL przed przemianowaniem lub usunięciem pliku;

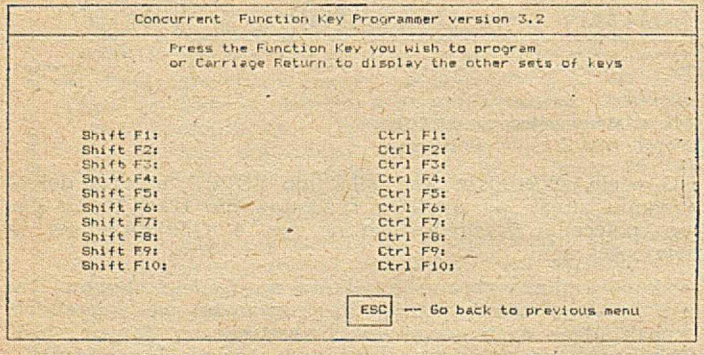
— definiowanie (rys. 5) i przypisywanie znaczenia klawiszom funkcyjnym; można definiować funkcje klawiszy F1—F10 oraz ich kombinacji z klawiszami ALT, CTRL i SHIFT, a także strzałki, HOME,. END, PGUP, PGDN, INSERT i DELETE (por. rys. 6—8 otrzymane po wybraniu klawisza F3 z rys. 5); ich znaczenie jest zapisywane zazwyczaj na pliku KEYS.PFK (dodatkowo można przedefiniować klawisze do przełączania okien);



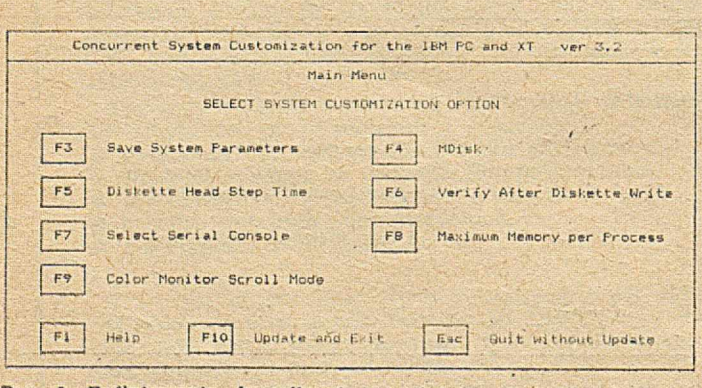
Rys. 6. Pierwszy podzbiór klawiszy funkcyjnych



Rys. 7. Drugi podzbiór klawiszy funkcyjnych

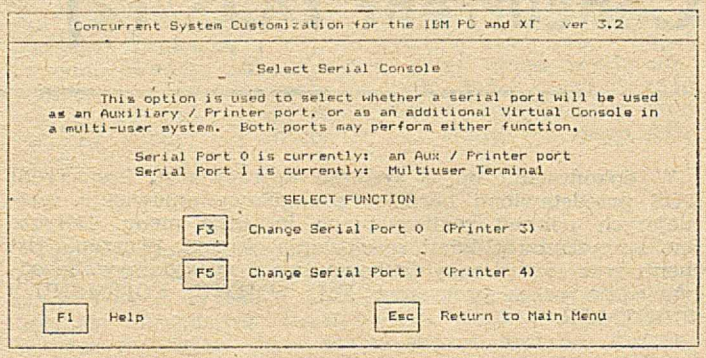


Rys. 8. Trzeci podzbiór klawiszy funkcyjnych

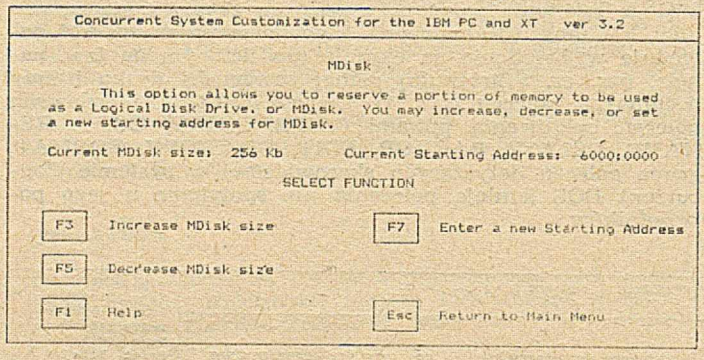


Rys. 9. Definiowanie domyślnych parametrów sprzętowych

— definiowanie domyślnych parametrów sprzętowych (rys. 9), jak czas przejścia głowicy dyskowej ze ścieżki na ścieżkę, przyporządkowanie portów szeregowych (drukarce lub dodatkowej konsoli, rys. 10), tryb przeglądania obrazu monitora kolorowego, adres i pojemność dysku elektronicznego (rys. 11), weryfikacja zapisu na dyskietce i maksymalna wielkość pamięci przydzielona programowi.



Rys. 10. Przyporządkowanie portów szeregowych



Rys. 11. Definiowanie adresu i pojemności dysku elektronicznego

OPEROWANIE OKNAMI — WMENU

Podobnie jak polecenia operowania plikami, wszystkie polecenia operowania oknami (wymienione w pierwszej części artykułu) można wydawać w sposób czysto tekstowy, bez użycia menu, lub posługując się techniką menu, do czego służy program WMENU.

Polecenie WINDOW VIEW powoduje wyświetlenie aktualnych parametrów wszystkich okien w postaci przedstawionej w tabeli. Poszczególne parametry mają następujące znaczenie:

- No. — numer okna,
- PRow — fizyczne położenie pierwszego wiersza okna na ekranie (ang. physical row),
- PCol — fizyczne położenie pierwszej kolumny okna na ekranie (ang. physical column),
- NRows — liczba wierszy okna,
- NCols — liczba kolumn okna,
- VRow, VCol — wirtualne położenie okna, tj. współrzędne punktu określającego umiejscowienie okna względem pełnego ekranu,
- Tracking — śledzenie ruchu kursora poza oknem,
- Display — rodzaj użytego monitora, monochromatyczny (B/W) lub kolorowy (Color); można korzystać z dwóch monitorów jednocześnie,
- FGColor — kolor tekstu w danym oknie (zależny od rodzaju monitora, tj. parametru Display),
- BGColor — kolor tła.

Przykładowo, zmiana parametrów okna nr 2 może mieć postać:

```
WINDOW CHANGE NUMBER=2 PR=12 PC=10 NR=10 NC=40
```

Standardowe parametry czterech okien (przykład)

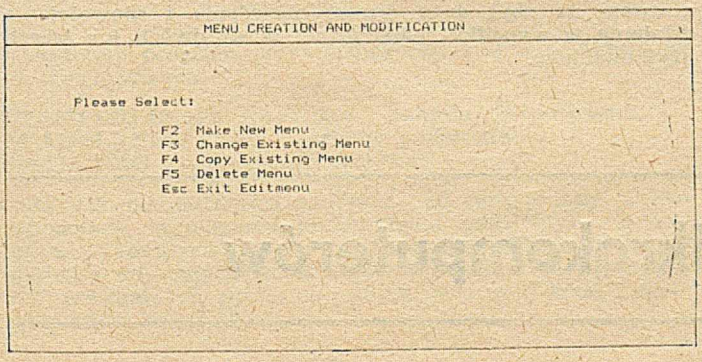
No.	PRow	PCol	NRows	NCols	VRow	VCol	Tracking	Display	FGColor	BGColor
1	1	1	24	80	1	1	Row	B/W	White	Black
2	12	16	10	40	1	1	Row	B/W	White	Black
3										
4										

Najłatwiejszym jednak sposobem pracy z oknami jest użycie list menu. Zamiast polecenia WINDOW z opcją CHANGE i tekstowym wyspecyfikowaniem odpowiednich parametrów, można użyć menu wyświetlanego na ekranie i wybrać odpowiednie parametry jednym z poleceń menu niższego poziomu (ang. submenu): SIZE (służy do określenia rozmiaru okna, NRows, NCols), PLACE (określa położenie okna, Prow itd.), TRACKING, DISPLAY lub COLOR. Wszystkie te zmiany mogą być zarejestrowane poleceniem WINDOW WRITE na pliku o nazwie WSETUP.BAT po to, aby przy ponownym inicjowaniu systemu zachować dotychczasowe parametry okien.

Warto też pamiętać, że każde okno może mieć własny plik wsadowy STARTUP, odgrywający rolę pliku AUTOEXEC.BAT w systemie operacyjnym PC-DOS. Pliki STARTUP1.BAT, STARTUP2.BAT itd., zawierają polecenia wykonywane przy każdorazowym inicjowaniu systemu Concurrent DOS.

TWORZENIE LIST MENU — EDITMENU

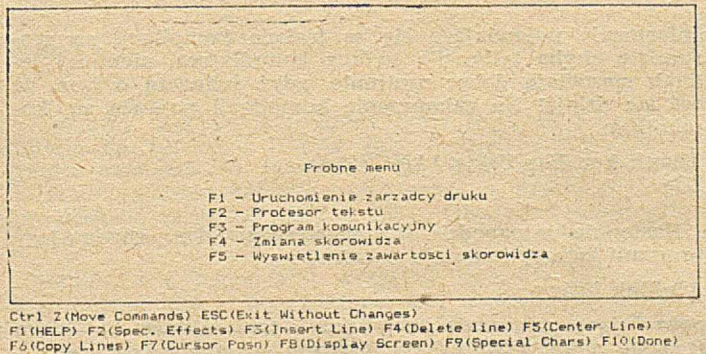
Pliki STARTUP i inne pliki poleceń mogą być utworzone tak, aby umożliwiały wybór kolejnych poleceń z listy menu. Do tworzenia tych list służy specjalne polecenie EDITMENU, używane — oczywiście — w trybie menu.



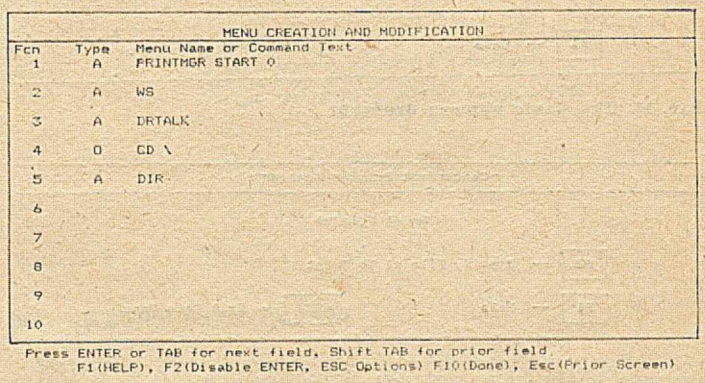
Rys. 12. Menu polecenia EDITMENU

Przykład typowego menu polecenia EDITMENU przedstawiono na rys. 12. Umożliwia ono utworzenie nowego menu (F2), zmianę (F3), skopiowanie (F4) i usunięcie (F5) istniejącego menu. Tworząc nowe menu należy określić:

- przypisanie poleceń poszczególnym klawiszom funkcyjnym w tym menu za pomocą specjalnego formularza (ang. assign function keys screen),
- tekst objaśnień wyświetlanych na ekranie w wypadku korzystania z tego menu, przy użyciu obszaru o wielkości ekranu (rys. 13).



Rys. 13. Tekst objaśnień utworzonego menu (ang. menu editing screen)



Rys. 14. Przypisanie poleceń klawiszom funkcyjnym (ang. assign function key screen)

Redagowanie tekstu objaśnień (ang. menu editing screen) odbywa się za pomocą dość typowego edytora z dobrze opracowanym samouczkiem, zawierającego m.in. takie funkcje, jak: wstawianie, usuwanie, centrowanie i kopiowanie wierszy, efekty specjalne (odwracanie obrazu na negatyw, podkreślanie, rozjaśnianie, migotanie itp.), znaki specjalne (graficzne, sterujące, nieangielskie itp.) i inne. Przypisanie poleceń klawiszom funkcyjnym dokonuje się na standardowym formularzu (rys. 14) podzielonym na następujące kolumny:

- numer klawisza funkcyjnego (Fcn),
- rodzaj polecenia (Type), np. A oznacza pełną treść polecenia, O wymaga przy wykonaniu podania parametrów polecenia i wciśnięcia klawisza ENTER, M oznacza nazwę nowego menu niższego poziomu (można dowolnie łączyć listy menu),
- treść polecenia w postaci wykonywalnej lub nazwa listy menu niższego poziomu (Menu Name or Command Text). Dla doświadczonych użytkowników przewidziano możliwość dodawania opcji do list menu (funkcja F2 z rys. 14).

W przykładzie przedstawionym na rys. 13 i 14 poszczególnym klawiszom funkcyjnym F1—F5 odpowiadają następujące funkcje:
— uruchomienie zarządcy druku, PRINTMGR START,
— zainstalowanie procesora tekstu WORDSTAR,
— załadowanie programu komunikacyjnego DRTALK,
— przejście do skorowidza o nazwie podanej z klawiatury jako parametr polecenia CD,
— wyświetlenie zawartości bieżącego skorowidza, DIR.

Przygotowane w ten sposób listy menu można używać posługując się poleceniem RUNMENU, np.:

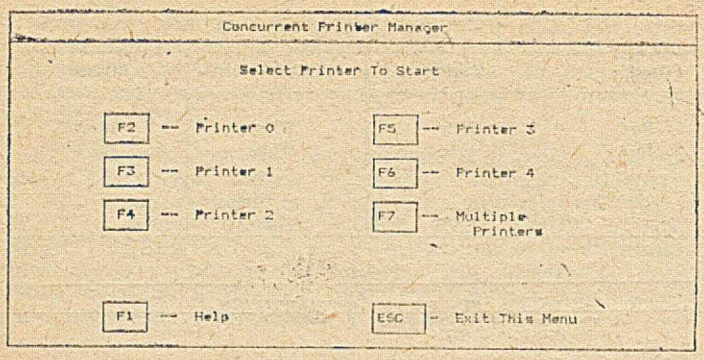
```
RUNMENU NOWEMENU.DAT
```

ZARZĄDZANIE DRUKIEM — PRINTMGR

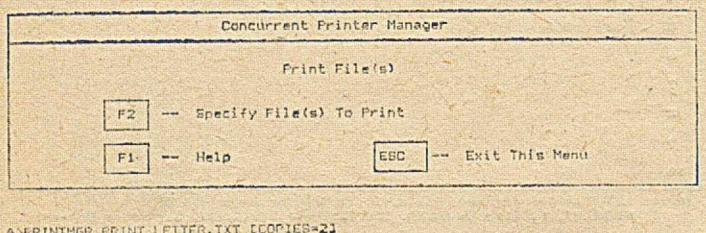
Innym ważnym programem możliwym do wykonania techniką menu jest zarządca druku, PRINTMGR (ang. printer manager). Z głównego menu zarządcy druku można wykonać m.in. następujące czynności:

F2 — określić numer drukarki (rys. 15),
F3 — podać nazwy plików do drukowania i parametry wydruku, opcje (rys. 16),
F4 — wyświetlić aktualny stan kolejki do drukowania,
F5 — zatrzymać proces drukowania bez możliwości wznowienia,
F6 — wstrzymać drukowanie z zachowaniem aktualnego stanu kolejki.

Wszystkie polecenia realizowane techniką menu mają swoje odpowiedniki w trybie tekstowym.



Rys. 15. Określanie numeru drukarki




```
A>PRINTMGR PRINT LETTER.TXT [COPIES=2]
```

Rys. 16. Menu opcji drukowania zawartości pliku

Parametry wydruku można dobierać następująco:

COPIES=n — liczba kopii (od 1 do 250, domyślnie n=1),

TAB=n — położenie tabulacji (domyślnie n=8),

MARGIN=n — położenie lewego marginesu (domyślnie n=8),

NOFEED — usunięcie domyślnego wysuwu wiersza przed każdą nową kopią,

PRINTER=n — numer drukarki (domyślnie n=0, tzn. LST:),

FORMAT — wprowadzenie odstępów między stronami tekstu (domyślnie — brak odstępów),

FORMSIZE=n — długość strony papieru (domyślnie n=66),

PAGESIZE=n — liczba wierszy na stronie (domyślnie n=66),

NUMBER — numeracja stron (domyślnie — brak numeracji).

* * *

Oprócz opisanych programów, w systemie Concurrent DOS mogą działać jeszcze inne, o bardziej specyficznych funkcjach, jak np. CARDFILE — pełniący rolę książki adresowej, lub DRTALK — program komunikacyjny, których można używać w technice menu z wykorzystaniem okien, lecz brak miejsca nie pozwala na ich przedstawienie w tym artykule.

Warto też dodać, że Concurrent DOS ma bardzo rozbudowany samouczek elektroniczny, HELP, który można uzupełniać stosownie do własnych potrzeb, używając opcji HELP EXTRACT. Jednym z ważniejszych programów, na który należy zwrócić uwagę, jest SYSDISK, służący do określania całego dysku lub pojedynczego skorowidza, jako domyślnego obszaru do poszukiwania plików. W systemie Concurrent DOS, wykorzystywanym jako wieloużytkowy, można również prowadzić ochronę plików, aby ustrzec je przed nieupoważnionym dostępem, czego nie ma w systemie PC-DOS.

Oceniając Concurrent DOS należy podkreślić, że ma on znaczną przewagę nad systemem operacyjnym PC-DOS ze względu na możliwość używania okien i operowanie techniką menu. Pozorna zaleta systemu Concurrent DOS, polegająca na wieloużytkowości, okazała się dość iluzoryczna z uwagi na zbyt duży wzrost obciążenia i spowolnienie działania w wypadku jednoczesnej pracy dwóch lub trzech użytkowników.

Kompilator Ady dla mikrokomputerów

W artykule opisano kompilator Janus/Ada firmy RR Software, dla procesora 8088, pracujący pod kontrolą systemu operacyjnego MS-DOS. Chociaż Janus/Ada nie jest kompilatorem pełnej Ady, to opracowanie takiej wersji jest w planach firmy. W obecnej postaci kompilator jest narzędziem, które można wykorzystać do zapoznania się z Adą; należy jednak pamiętać, że nie jest to pełna implementacja języka.

PORÓWNANIE Z PEŁNĄ ADĄ

Jakie są różnice między pełną Adą a jej częściową implementacją? W większości są to różnice mające charakter kosmetyczny, ale w jednym wypadku jest to różnica istotna: Janus/Ada nie przewiduje oddzielnie kompilowanych podprogramów. Każdy podprogram musi być umieszczony w pakiecie i kompilowany jako pakiet. W Janus/Adzie pojęcie programu głównego jest bliższe pojęciu pakietu, tzn. to, co projektanci Ady uważali za część inicjalizacyjną pakietu, staje się programem głównym. Nie jest to jednak duża wada. Byłoby dużo gorzej, gdyby Janus/Ada miał pojedynczą procedurę jako program główny. W tym stanie rzeczy Janus/Ada zapewnia odrębną kompilację specyfikacji pakietu i jego ciała. Pakiet może zawierać kilka podprogramów, jednakże w podanych dalej przykładach na jeden pakiet przypada tylko jeden podprogram. Omawiana wersja Janus/Ady nie ma też wyjątków ani agregatów.

Najkrótszy program Janus/Ady przedstawiono na wydruku 1.

Aby wykonać kompilację, program źródłowy umieszcza się, na przykład, na dysku B, a kompilator na dysku A.

```
-------------------------------
-- TINY.PKG
-- nazwa pakietu w Janus/Adzie
-- musi byc taka jak nazwa pliku;
-- najkrotszy program Janus/Ady
package body TINY is
-- program glowny
begin
   -- nie ma zadnych dzialan
   -- w programie glownym
   null;
end TINY;
-------------------------------
```

Wydruk 1

Następnie na dysk A przesyła się polecenie:

A>janus b:tiny

Informacje pojawiające się w trakcie kompilacji są zrozumiałe chyba tylko dla twórcy kompilatora, niemniej jednak sprawiają dobre wrażenie, gdyż świadczą o tym, że coś się dzieje! Na zakończenie kompilacji pojawia się komunikat:

Thank You For Using JANUS/Ada
A>

Następnie w napędzie A umieszcza się dysk zawierający program łączący i wydaje polecenie:

A>jlink b:tiny

powodujące połączenie programu i zapisanie go na dysku B jako pliku TINY.COM. Program wykonuje się poleceniem:

A>B:tiny

W celu przyspieszenia procesu kompilacji zaleca się umieszczenie kompilatora na dysku elektronicznym (ang. RAM-disk). Jeśli program łączący może pozostać na dyskietce w napędzie A, to dobrze jest utworzyć plik wsadowy, zawierający polecenia kompilatora, łączenia i wykonania.

Następna różnica w stosunku do pełnej Ady polega na tym, że nazwa pakietu musi odpowiadać nazwie pliku, na którym pakiet został zapisany. W systemie MS-DOS powoduje to ograniczenie nazwy pakietów do ośmiu znaków.

```
-- EXAMPLE.LIB
-- krótki program Ady
-- z podprogramem w pakiecie;
-- w Janus/Adzie specyfikacja
-- znajduje się na pliku lib
package EXAMPLE is
   -- specyfikacja podprogramu
   procedure DO_NOTHING;
end EXAMPLE;
```

```
-- EXAMPLE.PKG
-- krótki program Ady
-- z podprogramem w pakiecie
package body EXAMPLE is
   -- implementacja podprogramu
   procedure DO_NOTHING is
   begin
      null;
   end DO_NOTHING;
end EXAMPLE;
```

```
-- SMALL.PKG
-- program główny
with EXAMPLE;
-- program główny wykorzystuje
-- podprogramy z pakietu
package body SMALL is -- program główny
   -- użycie musi wystąpić
   -- w części deklaracyjnej
   use EXAMPLE;
   -- wykonywana część programu głównego
begin
   DO_NOTHING;
end SMALL;
```

Wydruk 2

Następny program przykładowy (wydruk 2) znajduje się na trzech plikach:

EXAMPLE.LIB dla specyfikacji pakietu podprogramu,
EXAMPLE.PKG dla ciała pakietu podprogramu,
SMALL.PKG dla programu głównego.

Pakiety nie odbiegają specjalnie od normy Ady. Prawdopodobnie, wymaganie umieszczania oddzielnych jednostek w oddzielnych plikach nie jest złym pomysłem w odniesieniu do małych komputerów. Powoduje to ograniczenie ewentualnych zmian do małych jednostek kodu i skrócenie czasu kompilacji.

```
-- HELLO1.PKG
-- powitalny komunikat w Adzie
package body HELLO1 is
begin
   PUT ("Dzień dobry!");
   NEW_LINE;
end HELLO1;
```

Wydruk 3

```
-- HELLO2.PKG
-- przywitanie w innej postaci
package body HELLO2 is
begin
   PUT ("Dzień");
   PUT (" ");
   PUT ("dobry");
   PUT ("!");
   NEW_LINE;
end HELLO2;
```

Wydruk 4

Programy wyświetlające komunikaty (wydruk 3 i 4) wyglądają na zgodne z normą Ady i działają prawidłowo. Trzeba pamiętać, że Janus/Ada nie wykorzystuje pakietu TEXT_IO, zamiast którego wbudowano do kompilatora różne postacie procedur GET i PUT. Obydwie wersje programu wyświetlają ten sam komunikat: Dzień dobry!

Janus/Ada wymaga, aby wszystkie zmienne były zadeklarowane i miały typ zdefiniowany pierwotnie. Główna różnica między programem na wydruku 5 a przykładem zgodnym z normą Ady wynika stąd, że procedury wejścia i wyjścia nie wymagają konkretyzacji. Stosując występujące w Janus/Adzie podtypy, można ten sam program zapisać w postaci przedstawionej na wydruku 6.

```
-- VAR1.PKG
-- wprowadzenie zmiennych
package body VAR1 is
   -- deklaracja zmiennej całkowitej
   LATA:INTEGER;
begin
   LATA := 40;
   PUT ("W tym roku Jan ukończył");
   PUT (LATA,2);
   PUT ("lat.");
   NEW_LINE;
end VAR1;
```

Wydruk 5

```
-- VAR2.PKG
-- wprowadzenie zmiennych i podtypów
package body VAR2 is
subtype TYP_LATA is INTEGER range 0..99;
-- deklaracja zmiennej typu TYP_LATA
   LATA : TYP_LATA;
begin
   LATA := 40;
   PUT("W tym roku Jan ukończył");
   PUT(LATA,2);
   PUT("lat.");
   NEW_LINE;
end VAR2;
```

Wydruk 6

STRUKTURY STEROWANIA

Struktury sterowania w Janus/Adzie dokładnie odpowiadają normie Ady. Przykłady z wydruków 7—9 należało umieścić w ciele pakietu, eliminując TEXT_IO. Pierwszy program ilustruje pętlę **while.** Na wydruku 8 przedstawiono inną wersję instrukcji **while,** wykorzystującą atrybut SUCC do inkrementacji zmiennej sterującej pętli. Ostatni przykład ilustruje instrukcję **for** (sterowanie przez odliczanie).

```
-- WHILE1.ADA
-- pętla while
package body WHILE1 is
   subtype TYP_LICZBY is INTEGER range 1..5;
   LICZBA : TYP_LICZBY;
begin
   LICZBA := 1;
   while LICZBA <= 4 loop
      PUT (LICZBA*10,2);
      NEW_LINE;
      LICZBA := LICZBA+1;
   end loop;
end WHILE1;
```

Wydruk 7

```
-- WHILE2.PKG
-- pętla while
package body WHILE2 is
   subtype TYP_LICZBY is INTEGER range 1..5;
   LICZBA : TYP_LICZBY;
begin
   LICZBA := 1;
   while LICZBA <= 4 loop
      PUT (LICZBA*10,2);
      NEW_LINE;
      LICZBA := TYP_LICZBY'SUCC(LICZBA);
   end loop;
end WHILE2;
```

Wydruk 8

```
-- FOR1.PKG
-- instrukcja for
package body FOR1 is
begin
   for LICZBA in 1..4 loop
      PUT (LICZBA*10,2);
      NEW_LINE;
   end loop;
end FOR1;
```

Wydruk 9

```
-- COPY1.ADA
-- kopiowanie wejścia na wyjście
with IO, UTIL;
package body COPY1 is
   use IO,UTIL;
   C : CHARACTER;
begin
   while not END_OF_FILE(STANDARD
         _INPUT()) loop
      GET(C);
      PUT(C);
   end loop;
end COPY1;
```

Wydruk 10

W Janus/Adzie procedury GET i PUT różnią się semantyką ze względu na odmienny sposób wykonywania operacji wprowadzania i wyprowadzania. Procedura GET czyta znak końca linii, co ułatwia zrozumienie działania programu kopiującego (wydruk 10). Dziwnie wyglądająca konstrukcja:

END_OF_FILE(STANDARD_INPUT())

stanowi pozostałość jednej z wcześniejszych wersji Ady — przed normalizacją. Jako znaku EOF zatrzymującego działanie programu nie używa się CTRL-Z lecz CTRL-C. Janus/Ada nie ma wyjątków, nie można więc tworzyć różnych wersji programu. Umieszczone w tym programie pakiety IO i UTIL zawierają funkcje END_OF_FILE i END_OF_LINE. Razem z kompilatorem Janus/Ada dostarczone są też inne pożyteczne funkcje i procedury wejścia-wyjścia, zawarte w tych dwóch pakietach.

```
-----------------------------------
-- IF1.PKG
-- ilustracja instrukcji if i else
package body IF1 is
   ODPOWIED2 : CHARACTER;
begin
   PUT ("Czy lubisz już Adę?");
   NEW_LINE;
   PUT ("Wciśnij t, jeśli tak, n, jeśli nie:");
   NEW_LINE;
   GET (ODPOWIED2);
   if ODPOWIED2='t' or ODPOWIED2='T' then
      PUT ("Miła wiadomość!");
   else
      PUT ("Miejmy nadzieję,
           że zmienisz zdanie.");
   end if;
end IF1;
-----------------------------------
```

Wydruk 11

```
-----------------------------------
-- GREETING.LIB
-- przykład instrukcji elsif
package GREETING is
   procedure GREET;
end GREETING;
-----------------------------------
-- GREETING.PKG
package body GREETING is
   procedure GREET is
   begin
      PUT ("Czy lubisz już Adę?");
      NEW_LINE;
      PUT ("Wciśnij t jeśli tak, n jeśli nie:");
      NEW_LINE;
   end GREET;
end GREETING;
-----------------------------------
-- ELSIF1.PKG
-- ilustracja użycia elsif
with GREETING;
package body ELSIF1 is
   -- ilustracja użycia if, else i elsif
   use GREETING;
   ODPOWIED2 : CHARACTER;
begin
   -- przywitanie użytkownika
   GREET;
   GET (ODPOWIED2);
   if ODPOWIED2='t' or ODPOWIED2='T' then
      PUT ("Miła wiadomość!");
      NEW_LINE;
   elsif ODPOWIED2='n' or ODPOWIED2='N'
         then PUT ("Bardzo nam przykro.");
      NEW_LINE;
   else
      PUT ("Nie rozumiem ");
      PUT ("odpowiedzi");
      NEW_LINE;
   end if;
end ELSIF1;
-----------------------------------
```

Wydruk 12

```
-----------------------------------
-- GREETS.LIB
-- specyfikacja pozdrowień
package body GREETS is
   procedure GREET;
end GREETS;
-----------------------------------
-- greets.pkg
-- IMPLEMENTACJA POZDROWIEŃ
package body GREETS is
   procedure GREET is
   begin
      PUT ("Wciśnij dowolny klawisz");
      NEW_LINE;
      PUT ("Zobaczymy czy odgadniesz literę");
      NEW_LINE;
      PUT ("Jeśli się znudzisz, wciśnij CTRL-C");
      NEW_LINE;
   end GREET;
end GREETS;
-----------------------------------
-- EXIT1.PKG
-- przykład instrukcji if i exit
with IO, UTIL, GREETS;
package body EXIT1 is
   use IO, UTIL, GREETS;
   C : CHARACTER;
begin
GREET; -- wyświetlenie pozdrowienia
   while not END_OF_FILE(STANDARD
                 _INPUT()) loop
      GET (C);
      if C='e' then
         PUT ("--Zgadłeś! Brawo!");
         NEW_LINE;
         exit;
      end if;
   end loop;
end EXIT1;
-----------------------------------
```

Wydruk 13

Instrukcja if nie różni się od opisanej w normie, ilustrują to przykłady wydruków 11 i 12. Różnica między tymi programami wynika głównie z faktu utworzenia oddzielnego pakietu GREET, co pociąga za sobą konieczność rozmieszczenia programu na trzech plikach (wydruk 12).

```
-----------------------------------
-- EXIT2.PKG
-- pokazanie sposobu eliminowania
-- instrukcji exit
with GREETS, IO, UTIL;
package body EXIT2 is
   use GREETS, IO, UTIL;
   -- inicjalizacja C na znak pusty
   C : CHARACTER:=' ';
begin
-- wyświetlenie pozdrowienia
   GREET;
   while C/='e' and not END_OF_FILE
     (STANDARD_INPUT()) loop
      GET (C);
      if C='e' then
         PUT ("Zgadłeś! Brawo!");
         NEW_LINE;
      end if;
   end loop;
end EXIT2;
-----------------------------------
```

Wydruk 14

Instrukcja **exit** jest również taka sama jak w normie. W poniższym przykładzie (wydruk 13) do opuszczenia programu wykorzystano konstrukcję END_OF_FILE. Choć znak CTRL-C jest niekoniecznie prawidłowym sygnałem końca pliku, ale zezwala prawidłowo opuścić program. W innej wersji przykładowego programu z instrukcją **exit** użyto ponownie pakietu IO i UTIL (wydruk 14).

OPEROWANIE TEKSTAMI

Następny program (wydruk 15) kopiuje znaki, aż do wystąpienia znacznika końca pliku. Funkcja END_OF_FILE jest zdefiniowana w pakiecie IO, a funkcje wejścia-wyjścia GET i PUT są wbudowane w kompilator. Funkcja STANDARD_INPUT znajduje się w pakiecie UTIL. Program umieszczono na trzech plikach:

COPY.LIB zawiera specyfikację pakietu nazywającego podprogram w specyfikacji procedury,

COPY.PKG stanowi implementację pakietu definiującego podprogram,

COPIES.PKG jest programem głównym wołającym podprogramy COPY.

```
-----------------------------------
-- COPY.LIB
-- kopiowanie wejścia na wyjście
package COPY is
   procedure KOPIUJ_WE_NA_WY;
end COPY;
-----------------------------------
-- COPY.PKG
-- kopiowanie wejścia na wyjście
with IO, UTIL;
package body COPY is
   procedure KOPIUJ_WE_NA_WY is
      use IO, UTIL;
      C : CHARACTER;
   begin
      while not END_OF_FILE
            (STANDARD_INPUT()) loop
         GET (C);
         PUT (C);
      end loop;
   end KOPIUJ_WE_NA_WY;
end COPY;
-----------------------------------
-- COPIES.PKG
-- program główny wywołujący procedurę
-- KOPIUJ_WE_NA_WY
with COPY;
package body COPIES is
   use COPY;
begin
   KOPIUJ_WE_NA_WY;
end COPIES;
-----------------------------------
```

Wydruk 15

Podprogram KOPIUJ_WE_NA_WY ma więcej sensu tutaj niż w przykładzie napisanym w pełnej Adzie, gdzie wymaga się specjalnej obsługi na przechowanie znaku CR (powrót karetki) i, jak się wydaje, jest wymagany wyjątek do obsługi znaku EOF. W naszym przykładzie prosta pętla **while** wykonuje się aż do osiągnięcia warunku końca pliku.

Następny zestaw pakietów (wydruk 16) wczytuje linię, dopóki nie wystąpi znacznik końca pliku lub powrotu karetki. Wczytywana linia jest później wyświetlana. Program główny znajduje się w pakiecie LINES.PKG, a podprogramy GET_MY_LINE i PUT_MY_LINE w pakiecie MYLINE.

```
-- MYLINE.LIB
-- wczytuje linie i wyświetla ją
package MYLINE is
   OGRANICZENIE : constant := 80;
   type TYP_NAPISOWY is array
     (1..OGRANICZENIE) of CHARACTER;
   subtype TYP_INDEKSOWY is INTEGER
      range 0..OGRANICZENIE;
   procedure GET_MY_LINE(LINIA : out
       TYP_NAPISOWY; DLUGOSC : out
       TYP_INDEKSOWY);
   procedure PUT_MY_LINE(LINIA : in
       TYP_NAPISOWY; DLUGOSC : in
       TYP_INDEKSOWY);
end MYLINE;
-----------------------------------
-- MYLINE.PKG
with IO, UTIL;
package body MYLINE is
   use IO, UTIL;
   procedure GET_MY_LINE(LINIA : out
       TYP_NAPISOWY; DLUGOSC : out
       TYP_INDEKSOWY) is
      C : CHARACTER;
   begin
      for I in 1..OGRANICZENIE loop
        GET (C);
        LINIA (I) := C;
        DLUGOSC := I;
      exit when END_OF_LINE(STANDARD_INPUT());
      exit when END_OF_FILE(STANDARD_INPUT());
      end loop;
   end GET_MY_LINE;
   procedure PUT_MY_LINE(LINIA : in TYP_NAPISOWY;
                 DLUGOSC : in TYP_INDEKSOWY) is
   begin
      for I in 1..DLUGOSC loop
        PUT (LINIA(I));
      end loop;
      NEW_LINE;
   end PUT_MY_LINE;
end MYLINE;
-----------------------------------
-- LINES.PKG
-- testowanie funkcji
with MYLINE;
package body LINES is
   use MYLINE;
   MOJA_LINIA : TYP_NAPISOWY;
   MOJA_DLUGOSC : TYP_INDEKSOWY;
begin
   GET_MY_LINE (MOJA_LINIA, MOJA_DLUGOSC);
   PUT_MY_LINE (MOJA_LINIA, MOJA_DLUGOSC);
end LINES;
-----------------------------------
```

Wydruk 16

Podczas kompilacji powyższego programu kompilator wywołany poleceniem:

A>janus b:myline.lib

najpierw kompiluje specyfikację pakietu MYLINE. Następnie, po powtórnym wywołaniu kompilatora poleceniem:

A>janus b:myline.pkg

kompiluje się ciało pakietu MYLINE. Trzecie wywołanie kompilatora:

A>janus b:lines.pkg

kompiluje program główny. Po komunikacie:

Thank You For Using JANUS/Ada

polecenie:

A>jlink b:lines

łączy segmenty programu i zapisuje go na dysku B jako plik LINES.COM. Program wykonuje się sekwencją poleceń:

A>time

A>b:lines

Oto normalne dane.

A>time

Pozwala to zarejestrować czas potrzebny na załadowanie i wykonanie programu (włączając czas potrzebny na wpisanie powyższego tekstu). Zajęło to mniej niż 20 sekund, z czego większą część pochłonęło ładowanie programu i pisanie komunikatu. Wydaje się, że program działa natychmiastowo, co jest naturalne dla tak prostych instrukcji.

Przykład ilustrujący w normie użycie instrukcji **case** również przepisano na Janus/Adę. Przy próbie utworzenia pętli zawierającej funkcję END_OF_LINE potrzebny znak trzeba wpisywać dwukrotnie. Okazuje się bowiem, że funkcja END_OF_LINE połyka znaki. Poza tym, przykład z wydruku 17 nie różni się wiele od wersji napisanej w pełnej Adzie.

Janus/Ada ma również typy wskaźnikowe. Główna różnica między programem przedstawionym na wydruku 18 a jego wersją w pełnej Adzie polega na tym, że kompilator Janus/Ada nie zezwala na alokację obiektu w czasie jego tworzenia. Takie założenie bardzo często występuje w kompilatorach częściowych. W rezultacie, zamiast jednej instrukcji jak w pełnej Adzie, są wymagane dwie. Inna różnica, w sposób niezbyt oczywisty wynikająca z programu, polega na tym, że napisy w Janus/Adzie nie mają stałej długości.

Istnieje możliwość uzyskania dostępu do pojedynczych znaków napisu przez wskazanie za pomocą zmiennej wskaźnikowej o postaci Z_NAPISU.all(i). Jednak przy dynamicznych napisach w Janus/Adzie nie można odwołać się do elementu napisu, zanim nie znajdzie się on w tym napisie. Z tego powodu w przykładowym programie nie można

```
-----------------------------------
-- HELP.LIB
-- ilustracja instrukcji case
with IO, UTIL;
package HELP is
   -- przesłanie poleceń wyświetlania
   use IO, UTIL;
   -- uzupełnienie brakującego znaku ASCII
   EOF : constant CHARACTER := CHARACTER'VAL(26);
   procedure GREET;
   -- przyjęcie odpowiedzi,
   -- filtrowanie odstępów, znaków tabulacji
   -- i powrotu karetki
   procedure GET_RESPONSE (C : out CHARACTER);
   -- wyświetlenie komunikatów
   type HELP_TOPIC_TYPE is (ADA_HELP, SYNTAX_HELP,
           LIBRARY_HELP, ERROR_HELP);
   procedure GET_HELP (HELP_TOPIC:
              HELP_TOPIC_TYPE);
end HELP;
-----------------------------------
-- HELP.PKG
with IO, UTIL;
package body HELP is
   use IO, UTIL;
   -- przesłanie poleceń wyświetlania
   procedure GREET is
   begin
      PUT ("Pomoc dla programujących w Adzie");
      NEW_LINE; NEW_LINE;
      PUT ("Naciśnij    Aby uzyskać informacje o");
      NEW_LINE;
      PUT ("------------------------------");
      NEW_LINE;
      PUT ("a           kompilatorze Ady");
      NEW_LINE;
      PUT ("s           składni Ady");
      NEW_LINE;
      PUT ("l           bibliotece Ady");
      NEW_LINE;
      PUT ("Twoje pytanie ?");
      NEW_LINE;
   end GREET;
   -- przyjęcie odpowiedzi
   -- filtrowanie odstępów
   procedure GET_RESPONSE (C : out CHARACTER) is
   begin
      C := ' ';
      while C = ' ' loop
        GET (C);
        if END_OF_FILE(STANDARD_INPUT())
        then
          C := EOF;
          exit;
        end if;
      end loop;
   end GET_RESPONSE;
   procedure GET_HELP (HELP_TOPIC :
             HELP_TOPIC_TYPE) is

      begin
        case HELP_TOPIC is
          when ADA_HELP =>
            PUT ("Polecenie ADA kompiluje
                 i wykonuje program");
            NEW_LINE;
          when SYNTAX_HELP =>
            PUT ("Polecenie ADAP analizuje
                 składnię programu.");
            NEW_LINE;
          when LIBRARY_HELP =>
            PUT ("Bibliotekę wykorzystuje się
                 stosując opcję /LIBFILE=[plik].");
            NEW_LINE
          when ERROR_HELP =>
            PUT ("Nie zrozumiano polecenia -
                 wykonaj program jeszcze raz.");
            NEW_LINE;
        end case;
   end GET_HELP;
end HELP;
-----------------------------------
-- MAIN.PKG
with HELP;
package body MAIN is
   use HELP;
   C : CHARACTER;
begin
   GREET;
   GET_RESPONSE (C);
   case C is
      when EOF => null;
      when 'a'|'A' => GET_HELP (ADA_HELP);
      when 's'|'S' => GET_HELP (SYNTAX_HELP);
      when 'l'|'L' => GET_HELP (LIBRARY_HELP);
      when others  => GET_HELP (ERROR_HELP);
   end case;
end MAIN;
-----------------------------------
```

Wydruk 17

```
-- POINT.PKG
-- typy wskaźnikowe - przykład
package body POINT is
   type WSKAZNIK_KOMUNIKATU is access STRING;
   KOMUNIKAT : WSKAZNIK_KOMUNIKATU;
begin
   KOMUNIKAT := new STRING;
   KOMUNIKAT.all := "Dzień dobry !";
   PUT (KOMUNIKAT.all);
end POINT;
```

Wydruk 18

użyć pętli **for** (wydruk 19). W tym wypadku procedura STRING_COPY składa się tylko z jednej instrukcji, więc wydaje się, że nie warto jej wyróżniać jako oddzielnego podprogramu.

```
-- MESSAGE.PKG
-- kopiowanie napisu typu wskaźnikowego
-- na napis
with STRLIB;
package body MESSAGE is
   use STRLIB;
   type WSKAZNIK_KOMUNIKATU is access STRING;
   PIERWOTNY : WSKAZNIK_KOMUNIKATU;
   KOPIA : STRING (20);
   procedure STRING_COPY
        (Z_NAPISU : in WSKAZNIK_KOMUNIKATU;
         NA_NAPIS : out STRING) is
   begin
      NA_NAPIS := Z_NAPISU.ALL;
   end STRING_COPY;
begin
   PIERWOTNY := new STRING;
   PIERWOTNY.all := "To się powtórzy";
   STRING_COPY (PIERWOTNY, KOPIA);
   PUT (PIERWOTNY.all);
   PUT (KOPIA);
end MESSAGE;
```

Wydruk 19

W programie WEEK (wydruk 20) do inicjalizacji tablicy DZIEŃ należałoby użyć agregatu. Ponieważ Janus/Ada nie ma tej konstrukcji, użyto instrukcji przypisania wartości poszczególnym elementom tablicy. Program ten można by dalej rozbudować wykorzystując atrybut IMAGE i wprowadzając obsługę wyjątków, jednak Janus/Ada nie zawiera takich mechanizmów.

```
-- WEEK.PKG
-- dni tygodnia
package body WEEK is
   type TYP_DNIA is array
      (INTEGER range 0..7) of STRING(9);
   DZIEN : TYP_DNIA;
   function DZIEN_TYGODNIA (LICZBA : INTEGER)
                            return STRING is
      D : STRING(9);
   begin
      if LICZBA < 1 or LICZBA > 7 then
         D := DZIEN(0);
      else
         D := DZIEN(LICZBA);
      end if;
      return D;
   end DZIEN_TYGODNIA;
begin
   DZIEN(0) := "błędne dane";
   DZIEN(1) := "poniedziałek";
   DZIEN(2) := "wtorek";
   DZIEN(3) := "środa";
   DZIEN(4) := "czwartek";
   DZIEN(5) := "piątek";
   DZIEN(6) := "sobota";
   DZIEN(7) := "niedziela";
   PUT ("Urodziłem się w "& DZIEN_TYGODNIA(4));
end WEEK;
```

Wydruk 20

```
-- LIST.PKG
-- ilustracja użycia tablicy rekordów
package body LIST is
   type TYP_LISTY is
      record
         NAZWA : STRING(13);
         NUMER : STRING(3);
      end record;
   type SPIS_TELEFONOW is
        array (1..3) of TYP_LISTY;
   TELEF : SPIS_TELEFONOW;
begin
   TELEF(1).NAZWA := "Zegarynka    ";
   TELEF(1).NUMER := "926";
   TELEF(2).NAZWA := "Straż pożarna";
   TELEF(2).NUMER := "998";
   TELEF(3).NAZWA := "Biuro zleceń ";
   TELEF(3).NUMER := "917";
   for I in 1..3 loop
      PUT (TELEF(I).NAZWA);
      PUT (TELEF(I).NUMER);
      NEW_LINE;
   end loop;
end LIST;
```

Wydruk 21

Ostatni program (wydruk 21) ilustruje korzystanie z rekordów. W ten sposób można częściowo zastąpić użycie agregatów pełnej Ady. Brak atrybutu RANGE powoduje, że w wypadku zmiany liczby elementów listy należy zmienić całą pętlę.

* * *

Te proste przykłady wykazują, że stopień podobieństwa Janus/Ady do pełnej Ady jest dosyć duży. Szczegółowa ocena powinna być jednak dokonana samodzielnie przez każdego użytkownika. Można mieć nadzieję, że następne wersje kompilatora będą coraz bliższe pełnej Adzie.

Zamieszczone przykłady nie ilustrują niestety diagnostyki komunikatów o błędach. Na przykład po opuszczeniu średnika w tekście programu kompilator pyta, czy użytkownik chce go wstawić i przejść do dalszych działań.

Przy nieprzestrzeganiu zawężeń system wykonawczy informuje, która linia i w którym podprogramie powoduje błąd, jak również jaki program nadrzędny powoduje wywołanie i w której linii to następuje.

Opracował MARIUSZ KUC
na podst. BYTE, No. 7, 1984

Konferencje

Techniki komputerowe w zarządzaniu produkcją

Oddział Wojewódzki Polskiego Towarzystwa Ekonomicznego w Bydgoszczy, wspólnie z Instytutem Badań Systemowych Polskiej Akademii Nauk oraz Zakładem Informatyki Stosowanej Akademii Techniczno-Rolniczej — organizują w dniach 22—24 września 1987 roku w Ciechocinku ogólnopolską konferencję naukową nt. **Techniki komputerowe w zarządzaniu produkcją — INFOPROD-87.**

Celem konferencji jest ocena zastosowań komputerów, minikomputerów i mikrokomputerów w zarządzaniu produkcją oraz przedyskutowanie i wytyczanie obszarów i metod dalszych ich wdrożeń, a w szczególności:

- zastosowanie komputera R-34 w zarządzaniu produkcją,
- zaprezentowanie przykładów praktycznych zastosowań minikomputerów SM-3 i MERA-9150 we współpracy z dużymi komputerami,
- prezentacja możliwości i wykorzystania mikrokomputerów IBM PC/XT i AT oraz MERA 660,
- prezentacja sprzętu i oprogramowania (wystawa sprzętu),
- przedyskutowanie koncepcji zmian w metodach sterowania produkcją przez szefów produkcji, z wykorzystaniem sprzętu komputerowego, mini- i mikrokomputerów.

Konferencja jest adresowana głównie do:

- dyrektorów produkcji i dyrektorów ekonomicznych przedsiębiorstw,
- szefów produkcji przedsiębiorstw,
- kierowników wydziałów produkcyjnych oraz służb:
  - kompletacyjnych i planowania produkcji,
  - gospodarki materiałowej,
  - ekonomicznych,
  - informatycznych.

W programie konferencji przewiduje się wygłoszenie kilkunastu referatów oraz przprowadzenie dyskusji panelowych.

Wszelkich informacji udzielają organizatorzy:

Polskie Towarzystwo Ekonomiczne
Oddział Wojewódzki
ul. Długa 34
85-034 Bydgoszcz

Systemy ekspertowe na mikrokomputery IBM PC

Programy sztucznej inteligencji wymagają dużych zasobów pamięciowych, z reguły większych od tych, którymi dysponują małe systemy komputerowe. Jednakże ze względu na olbrzymie sukcesy rynkowe komputerów osobistych, firmy wytwarzające takie programy zainteresowały się również systemami mikrokomputerowymi.

Większość produktów z zakresu sztucznej inteligencji, przeznaczonych dla komputerów osobistych, stanowi narzędzia wspomagające budowę systemów ekspertowych. Bardzo często są one mikrokomputerowymi wersjami (zwykle zrealizowanymi w języku C) systemów opracowanych pierwotnie na dużych komputerach z wykorzystaniem Lispu lub Prologu. Systemy tego rodzaju są zestawami uniwersalnych narzędzi, które w znacznym stopniu ułatwiają użytkownikowi komputera osobistego budowę systemu ekspertowego w określonej dziedzinie zastosowań, niezależnie od tego, czy jest to analiza echokardiogramów, czy też wyszukiwanie usterek systemu grzewczego.

Poniższy artykuł zawiera krótki przegląd bardziej znanych na rynku amerykańskim narzędzi wspomagających budowę systemów ekspertowych na komputerach klasy IBM PC. Uwzględniono również przykłady bezpośredniego wykorzystania technik sztucznej inteligencji oraz przykłady konkretnych zastosowań zrealizowanych za pomocą systemów wspomagających oraz rzadki przypadek połączenia systemu wspomagającego z konkretnym zastosowaniem. Jest to system Guru, który integruje w jednym środowisku narzędzia budowy systemów ekspertowych oraz narzędzia zarządzania przedsiębiorstwem (ang. business tools).

SYSTEMY WSPOMAGAJĄCE I ICH ZASTOSOWANIA

Firma Teknowledge Inc. oferuje programy wspomagające **M.1** i **S.1**. Program M.1 jest przeznaczony dla IBM PC, PC/XT, PC/AT oraz dla komputerów z nimi zgodnych i umożliwia budowę małych i średnich systemów ekspertowych zawierających nie więcej niż 1000 reguł. Program S.1 jest przeznaczony dla IBM RT/PC i umożliwia budowę większych i bardziej złożonych systemów. Firma wprowadzi wkrótce na rynek nową wersję programu S.1 z przeznaczeniem dla PC/AT.

Firma Level Five Research oferuje dwa systemy wspomagające, **Insight 2+** i **Insight 1+**, jak również system ekspertowy **Cocomo 1** zbudowany za pomocą Insight 2+. Insight 2+ jest narzędziem ułatwiającym integrację baz wiedzy, funkcji matematycznych oraz programów i urządzeń zewnętrznych. System wspomagający Insight 1+ uczy jednocześnie użytkownika, jak należy budować systemy ekspertowe. System ekspertowy Cocomo 1 pomaga ocenić nakład pracy, koszty i czas niezbędnych do realizacji projektów programistycznych. W systemie wykorzystano matematyczny model kosztów oprogramowania zaproponowany przez B. Boehma z firmy TRW.

System **RuleMaster** firmy Radian generuje automatycznie reguły oparte na przykładowych sytuacjach opisanych przez użytkownika. System posługuje się specjalnym językiem reguł o nazwie Radial. Reguły są definiowane za pomocą hierarchii drzew decyzyjnych, a więc system nie jest oparty na jednej dużej bazie reguł i osobnej maszynie wnioskującej. Małe zbiory reguł tworzące hierarchie łączy się za pomocą tradycyjnych metod, stosowanych w językach strukturalnych. System RuleMaster generuje na żądanie wersję systemu ekspertowego w języku C, co pozwala na łączenie systemów ekspertowych z innymi programami. Mikrokomputerowa wersja systemu RuleMaster posługuje się kompilatorem języka C.

Przykładem integracji pakietu wspomagającego budowę systemów ekspertowych i narzędzi zarządzania przedsiębiorstwem jest **Guru** firmy Micro Data Base Systems (MDBS). W skład programu wchodzi administrator zbioru reguł i maszyna wnioskująca. Narzędzia zarządzania przedsiębiorstwem obejmują elektroniczny arkusz obliczeniowy, obsługę bazy danych, zdalną komunikację i przetwarzanie tekstów. Guru jest rozszerzeniem systemu zarządzania bazą danych Knowledge Man/2 (tej samej firmy) o takie techniki sztucznej inteligencji jak wnioskowanie wstępujące i zstępujące (ang. forward, backward chaining). Guru pracuje w sieciach lokalnych IBM PC, Novell Netware i 3Com Ethernet.

KES II firmy Software Architecture and Engineering jest nową wersją systemu wspomagającego budowę systemów ekspertowych KES (ang. Knowledge Engineering System) tej samej firmy, znanego z realizacji na dużych komputerach. KES II, napisany w języku C, ma większe możliwości niż system KES.

Firma Human Edge Software oferuje system wspomagający **Expert Edge** oraz system prognozujący **Forcasting Edge**. Expert Edge zawiera w sobie maszynę wnioskującą. Wprowadzanie danych odbywa się za pomocą języka naturalnego lub też bezpośrednio z baz danych, elektronicznych arkuszy obliczeniowych i procesorów tekstowych. System przetwarza niepewne lub niekompletne informacje z wykorzystaniem rachunku prawdopodobieństwa i statystyki Bayesa. Ma możliwość uzasadniania podjętych decyzji oraz mechanizm kontekstowej generacji tekstów objaśniających działanie budowanego systemu ekspertowego. System Forcasting Edge umożliwia użytkownikowi, który nie zna technik prognozowania i statystyki, konstruowanie prognoz z zakresu zarządzania z wykorzystaniem metody Boxa-Jenkinsa.

KDS firmy KDS Corp. jest systemem generującym automatycznie tzw. moduły wiedzy, z których można następnie tworzyć większe systemy. Ten system posługuje się zarówno strategią wnioskowania wstępującego, jak i zstępującego. Umożliwia wykonywanie zewnętrznych programów oraz wczytywanie zewnętrznych danych. Do odtwarzania modułów wiedzy wygenerowanych przez różne instalacje systemu KDS jest niezbędny specjalny pakiet odtwarzający. Tańszy program odtwarzający, dołączony do konkretnego systemu KDS, pozwala odtwarzać moduły wiedzy wygenerowane tylko w ramach tego systemu.

Firma Airus oferuje dwa produkty, które są oparte na tzw. przetwarzaniu intuicyjnym. Pierwszy z nich, **Airus-A**, jest narzędziem programowania. Został napisany w języku C i wykorzystuje pewien algorytm sztucznej inteligencji, który zapewnia dostęp do lingwistycznych baz danych. Porównuje on słowa wejściowe ze słownikiem i udostępnia odpowiednie atrybuty niezbędne w dalszym przetwarzaniu. Atrybuty przypisane określonym znacznikom (ang. token) mogą zawierać kody, indeksy tablic lub pełne wskaźniki adresowe. Firma wykorzystała system Airus-A do budowy pakietu **AI:Typist**, procesora tekstowego sprawdzającego na bieżąco pisownię tworzonego tekstu. Procesor generuje dokumenty w postaci plików sformatowanych lub plików w standardzie ASCII.

INNE NARZĘDZIA

Firma IntelliCorp oferuje pakiet **KEE PC-Host**, który umożliwia przeniesienie i eksploatację na komputerach osobistych systemów ekspertowych, zbudowanych za pomocą firmowego systemu KEE (Knowledge Engineering Environment). Jest on dostępny na maszynach lispowych (tzn. maszynach o specjalnej architekturze dostosowanej do symbolicznego przetwarzania list). Komputery osobiste obsługują program dostępu do systemu KEE, a właściwy system ekspertowy (wygenerowany w Common Lispie) jest przetwarzany na komputerze głównym typu VAX. Sprzężenie graficzne systemu KEE zostało zrealizowane za pomocą języka C oraz pakietu graficznego GEM firmy Digital Research.

Może ono pracować na mikrokomputerach klasy IBM PC wyposażonych w 512 KB pamięci, port szeregowy, dwa napędy dysków elastycznych oraz system operacyjny MS-DOS w wersji 2.0 lub następnej. Komputer osobisty i komputer główny mogą być połączone lokalnie za pomocą standardowego łącza szeregowego lub za pomocą modemu telefonicznego. KEE PC-Host kosztuje 27 500 dolarów w wersji z maszyną główną, wliczając w to licencję na oprogramowanie, pięć dni konsultacji i rok nadzoru. Samo oprogramowanie kosztuje 15 000 dolarów, natomiast licencja na oprogramowanie dla jednego komputera osobistego kosztuje 500 dolarów.

Firma Gold Hill Computers oferuje **Golden Common Lisp (GCLisp)** oraz narzędzia wspomagające programowanie. GCLisp jest rozbudowanym środowiskiem programowania Common Lispu i zastosowań sztucznej inteligencji. Jest dostępny na IBM PC, PC/XT i PC/AT, wyposażonych w pamięć 512 KB, napęd dysku elastycznego z obustronnym zapisem o podwójnej gęstości oraz system operacyjny PC-DOS lub MS-DOS w wersji 2.0 lub następnej. Zaleca się zainstalowanie napędu dysku stałego lub drugiego napędu dysku elastycznego, jak również procesora zmiennoprzecinkowego i drukarki. Kompilator ze zbiorem narzędzi kosztuje 495 dolarów.

GCLisp 286 Developer, bogatsza wersja GCLispu, jest systemem wspomagającym realizację zastosowań sztucznej inteligencji na mikrokomputerach IBM PC/AT. Składa się z interpretera, kompilatora, edytora, programu uczącego oraz dokumentacji dostępnej w trybie on-line. Produkt wymaga co najmniej 2 MB pamięci, systemu operacyjnego PC-DOS w wersji 3.0 lub następnej, napędu dysku stałego oraz napędu dysku elastycznego z obustronnym zapisem o podwójnej lub poczwórnej gęstości. Firma zaleca eksploatację produktu na IBM PC/AT z pamięcią o pojemności co najmniej 3 MB, myszką oprogramowaną przez firmę Mouse Systems, adapterem graficznym o zwiększonej rozdzielczości i kolorowym monitorem. GCLisp 286 Developer kosztuje około 1000 dolarów.

Narzędziem uzupełniającym do wymienionych wyżej produktów jest **GCLRun,** za pomocą którego można tworzyć bezpośrednio wykonywalne wersje programów napisanych w GCLispie. GCLRun generuje pliki binarne na podstawie programów skompilowanych uprzednio przez kompilator GCLispu. GCLRun umożliwia dołączanie do GCLispu programów napisanych w innych językach programowania. Cena produktu waha się od 100 dolarów (zakup 5—99 kopii) do 5 dolarów (zakup ponad 50 000 kopii).

Oprócz narzędzi przeznaczonych dla komputerów IBM PC powstają też narzędzia dla innych komputerów osobistych. Przykładowo, firma Neuron Data Corp. zbudowała system wspomagający **Nexpert** dla komputerów Macintosh o pamięci 512 KB i komputerów Macintosh Plus. Nexpert akceptuje maksimum 500 reguł i daje możliwość wnioskowania z wykorzystaniem strategii wstępującej, zstępującej oraz mieszanej. Edytor bazy wiedzy składa się z edytora reguł, edytora kategorii oraz edytora tekstowego. Technika okien umożliwia przeglądanie danych, hipotez, reguł i kategorii zawartych w bazie wiedzy. Użytkownik może zmieniać wartości standardowych parametrów maszyny wnioskującej, klasyfikować i określać priorytety danych w zależności od kategorii, łączyć dane graficzne i tekstowe, drukować lub gromadzić raporty, jak również żądać od systemu uzasadnienia wyprowadzonych wniosków. Nexpert akceptuje pliki w formacie SYLK, a więc może wykorzystywać dane dostarczone przez elektroniczne arkusze obliczeniowe Excel firmy Microsoft.

Opracował **M. MACHURA**
na podst. **COMPUTER**, lipiec 1986

Zestawienie omówionych systemów

System	Sprzęt	Język implementacji	System operacyjny	Cena (dolary)	Producent
M.1	IBM PC, PC/XT, PC/AT i zgodne	C	MS-DOS, PC-DOS	5 000	Teknowledge Inc. 1850 Embarcadero Rd. PO Box 10119 Palo Alto, CA 94303
S.1	512 KB IBM RT/PC, 2 MB	C	MS-DOS	25 000	
Insight 2+	IBM PC, PC/XT, PC/AT i zgodne, 512 KB, zalecany dysk stały	Turbo Pascal	MS-DOS lub PC-DOS wersja 2.0 lub następne	485; 95 za wersję binarną	Level Five Research, Inc. 503 Fifth Ave. Indialantic, Fl 32903
Insight 1+	IBM PC, PC/XT, PC/AT i zgodne, 340 KB	Turbo Pascal	MS-DOS lub PC-DOS wersja 2.0 lub następne	95	
Cocomo 1	IBM PC, PC/XT, PC/AT i zgodne, 340 KB, dwa napędy dyskowe	Turbo Pascal	MS-DOS lub PC-DOS wersja 2.0 lub następne	100	
RuleMaster	IBM PC/XT, PC AT i zgodne, 640 KB, dysk stały	C	DOS	995	Radian Corp. 8501 Mo-Pac Blvd. PO Box 9948 Austin TX 78766
Guru	IBM PC i zgodne, 512 KB, dysk stały	C	PC-DOS, MS-DOS	2 995; 300 za wersję binarną	Micro Data Base Systems Inc. PO Box 248 Lafayette, IN 47902
KES II	IBM PC/AT, PC/XT i zgodne, 640 KB	C	MS-DOS	4 000	Software Architecture and Engineering, Inc. 1600 Wilson Blvd. Arlington, VA 22209
Expert Edge	IBM PC i zgodne, 256 KB (zalecane 512 KB), dwa napędy dyskowe	C	PC-DOS lub MS-DOS wersja 2.0 lub następne	795	Human Edge Software Corp. 1875 S. Grant St. San Mateo, CA 94402—2669
Forecasting Edge	IBM PC i zgodne, 256 KB	Pascal	PC-DOS lub MS-DOS wersja 2.0 lub następne	100	
KDS	IBM PC, PC/AT, PC/XT i zgodne, 512 KB (320 KB dla wersji binarnej), zalecane dwa napędy dyskowe	Język asemblera	PC-DOS	795 (+cena programów odtwarzających)	KDS Corp. 934 Hunter Rd. Wilmette, IL 60091
Airus-A	IBM PC i zgodne, 256 KB	Lattice Microsoft C	MS-DOS wersja 2.0 lub następne	150	Airus, Inc. 10200 S.W. Nimbus Ave., Suite G-5 Portland, OR 97223
AI: Typist	IBM PC i zgodne, 256 KB	C	MS-DOS wersja 2.0 lub następne	100	

Konferencje

Entity — Relationship Approach'86

„Jeżeli myślisz, że nigdy nie słyszałeś o Dijon, to spójrz na swój słoik musztardy — prawdopodobnie pochodzi z Dijon. A jeżeli przypadkiem posiadasz w swojej piwnicy butelkę burgundzkiego wina, to powinieneś wiedzieć, że Dijon jest stolicą tego sławnego regionu Francji". W ten sposób prezentowano Dijon jako miejsce kolejnej międzynarodowej konferencji nt. podejścia do modelowania konceptualnego, znanego pod angielską nazwą **Entity Relationship (ER) Approach** [1]). Tegoroczna konferencja, piąta z tego cyklu a pierwsza poza granicami USA, odbyła się w dniach 17—19 października 1986 r. Była finansowana m.in. przez ER Institute oraz IFIP WG-2 (bazy danych), a jej organizatorem, we współpracy z ACM i IEEE Computer Society, było francuskie towarzystwo informatyczne AFCET [2]). Główne tematy konferencji przedstawiono poniżej:

- projektowanie baz danych i systemów informacyjnych,
- modele danych oraz techniki modelowania danych,
- języki manipulacji danymi i sprzężenia użytkownika,
- dynamika i integralność baz danych,
- definicje formalne dla podejścia ER,
- wielonośnikowe bazy danych,
- systemy z bazą wiedzy,
- rozszerzenia dla systemów o specjalnym przeznaczeniu (np. do obliczeń inżynierskich, komputerowego wspomagania projektowania i produkcji, itp.),
- doświadczenia w szkoleniu i nauczaniu.

Podstawy podejścia ER zostały ustanowione w artykule Petera Chena opublikowanym w 1976 roku [3]). Od tego czasu datuje się jego rozwój i zastosowanie na całym świecie. Jest ono obecnie najpowszechniej stosowane przy projektowaniu baz danych do rzeczywistych zastosowań. Można również przypuszczać, że będzie ono wiodące w nadchodzącej postrelacyjnej erze w bazach danych.

[1]) W języku polskim podejście to określa się niekiedy skrótem OAR (obiekt-atrybut-relacja).

[2]) AFCET — Association Francaise pour la Cybernètique, Economique et Technique.

[3]) P. P. Chen: The entity relationship model — toward a unified view of data. ACM Trans. on Database Systems. Vol. 1, No. 1, pp. 9—36, 1976.

Obecna konferencja uświetniła dziesięciolecie opublikowania podstawowego artykułu Petera Chena. Autor wygłosił jeden z trzech specjalnych wykładów przygotowanych na konferencję, analizując w nim rezultaty ostatniej dekady w dziedzinie podejścia ER oraz przedstawiając przewidywania na następne 10 lat. Drugi z wykładów specjalnych, autorstwa P. Kinga z Uniwersytetu Londyńskiego, dotyczył procesu projektowania bazy danych, a trzeci — dwuczęściowy, przygotowany przez A. Rochfelda i H. Tardieu, francuskich liderów w dziedzinie stosowania podejścia ER — naświetlał metodologię i warsztat projektowania użytkowych systemów informacyjnych. Krótkie wprowadzenia do tych wykładów zamieszczono w tabelach.

W trakcie konferencji odbyło się 13 sesji roboczych, na których zaprezentowano cztery referaty przygotowane na zamówienie organizatorów oraz 21 prac, wyselekcjonowanych spośród 67 nadesłanych z 20 krajów (najwięcej z USA — 18 oraz z Francji — 10). W jednym z referatów zamówionych przedstawiono cele projektu ESPRIT w dziedzinie baz danych (projekt cia w zakresie wielonośnikowych (ang. multimedia) baz danych. Pozostałe dwa zamówione referaty miały charakter raportów z dwóch ostatnich konferencji nt. systemów obiektowych (ang. object-oriented systems) oraz systemów z bazą wiedzy (ang. knowledge based).

W referatach zakwalifikowanych na konferencję zaprezentowano szeroki wachlarz zagadnień obejmujący zarówno rezultaty teoretyczne, jak i zastosowania praktyczne oraz zaimplementowane systemy oparte na podejściu ER. Każdy z nadesłanych referatów był opiniowany przez trzech lub czterech recenzentów. Ocenie, w skali od 1 do 6 punktów, podlegały następujące aspekty prac: stopień powiązania z tematyką konferencji, oryginalność, znaczenie prezentowanych tez, jakość techniczna, czytelność i organizacja tekstu oraz użyteczność opracowania. Jednak mimo zastosowania tych kryteriów poziom artykułów był zróżnicowany. Warto wspomnieć, że w konkursie prac najwyższą ocenę recenzentów uzyskała (wspólnie z jedną

Wprowadzenie do wykładu 1 — P. P. Chen (Louisiana State University, USA): Podejście ER — minione dziesięć lat oraz następna dekada

Teorię podejścia ER zaprezentowano 10 lat temu. W minionej dekadzie obserwowano powstawanie wielu nowych rozszerzeń teoretycznych tego podejścia oraz jego praktycznych zastosowań. Teoria ta stała się integralną częścią wielu podręczników z zakresu analizy baz danych i systemów (ang. database and systems textbooks). Została także włączona do programów nauczania zalecanych zarówno przez ACM jak i przez IEEE. Jeżeli chodzi o stronę praktyczną, to podejście ER zostało pomyślnie użyte w analizie systemów i w przedsięwzięciach z dziedziny projektowania baz danych, realizowanych przez wiele organizacji gospodarczych.

W wykładzie oceniono główne elementy ewolucji podejścia ER w ciągu minionych 10 lat, zaprezentowano przypuszczenia dotyczące tego, co zdarzy się w następnej dekadzie, a także określono, co należy zrobić, aby dobrze wykorzystać rysujące się już obecnie przyszłe możliwości.

Wprowadzenie do wykładu 2 — P. J. H. King (Birkbeck College, University of London, Wielka Brytania): Proces projektowania bazy danych

System bazy danych o odpowiednio dużym rozmiarze i złożoności wymaga starannego projektowania zarówno na poziomie fizycznym, jak i logicznym. W celu modelowania interesującego fragmentu świata rzeczywistego, na poziomie logicznym powinna być użyta określona technika projektowania; stowarzyszonego modelu procesu można użyć do sprawdzenia, czy zbudowany model logiczny nadaje się do opisu działania podstawowego procesu i zapytań.

Wykład podsumowuje i wyjaśnia powyższe podejście i związane z nim techniki odwzorowywania, których celem jest uzyskiwanie projektu fizycznego z projektu logicznego.

Wprowadzenie do części 1 wykładu 3 — A. Rochfeld (konsultant, Paryż, Francja): MERISE — metodologia projektowania i rozwoju systemu informacyjnego

Celem tej części wykładu jest prezentacja metodologii MERISE. Rozpatrzono następujące zagadnienia:

1) podstawy systemu,

2) aspekt modelowania — szczegółowo omówiono poziom abstrakcyjny oraz potrzebę podejścia trzypoziomowego; reguły formalne dla definiowania poziomu konceptualnego przedstawiono w powiązaniu z podejściem ER oraz sieciami Petriego; prześledzono reguły translacji do innych poziomów,

3) cykl istnienia MERISE,

4) metodologie projektowania i rozszerzony model ER.

Wprowadzenie do części 2 wykładu 3 — H. Tardieu (SEMA-METRA, Paryż, Francja): Warsztat projektowania systemów informacyjnych — udział podejścia ER w zarządzaniu obiektami

Warsztat projektanta systemów informacyjnych składa się ze zbioru narzędzi umożliwiających korzystanie z pewnego rodzaju magazynu, w którym przechowywane są różne obiekty opisujące pojęcia używane w wybranej technice modelowania. Podejście ER jest szeroko używane do modelowania struktury danych systemu zarządzania obiektami (ang. object management system). Korzystając z formalizmu podejścia ER przedstawiono porównanie różnych podejść: IRDS (ISO), PCTE (ESPRIT), S1 (CONCERTO), IEW (Knowledgeware), warsztat METRA (SEMA-METRA).

Zaprezentowano niezbędne uzupełnienia w podejściu ER, pozwalające spełnić wymagania dotyczące zarządzania obiektami w warsztacie projektanta systemów informacyjnych oraz porównano je ze zbieżnym procesem budowy oprogramowania przy użyciu języków obiektowych (ang. object-oriented languages). Podano następnie wszechstronny opis różnych środowisk (specyfikacji, projektowania, dokumentacji, zarządzania projektem, zarządzania konfiguracją itp.) w terminach modelu ER. Na zakończenie przedstawiono zalety (elastyczność i spójność) oraz wady (koszt działania i rozwoju) metody.

z prac amerykańskich) praca polskich autorów: Kazimierza Subiety i Marka Missali (obaj z Instytutu Podstaw Informatyki PAN w Warszawie) pt. „Semantyka języków zapytań dla modelu ER".

W czasie trwania konferencji odbyło się także kilka atrakcyjnych imprez dodatkowych. Jedną z nich była prezentacja działających produktów i prototypów programistycznych. Pozostałe, takie jak koktail w siedzibie władz miejskich Dijon, bankiet w historycznym zamku, pięknie położonym wśród burgundzkich winnic, wycieczka do Beaune na wielką aukcyjną sprzedaż win na cele dobroczynne oraz udział w uroczystości degustacji win, miały już charakter rozrywkowy.

W zgodnym odczuciu uczestników, konferencja zakończyła się dużym sukcesem. Pełne teksty wykładów, referatów zamówionych i przyjętych na konferencję oraz inne materiały konferencyjne będą opublikowane w formie książkowej nakładem wydawnictwa North-Holland.

WIKTOR RZECZKOWSKI

Kto jest kim w IFIP

Heinz Zemanek

Prof. dr Heinz Zemanek, członek honorowy, a niegdyś prezydent IFIP, urodził się w 1920 roku w Wiedniu. Tu również przeszedł swoją edukację od szkoły podstawowej do uniwersytetu. Po odbyciu służby wojskowej rozpoczął pracę w spółce elektrycznej, lecz szybko, bo w 1947 roku, powrócił do Uniwersytetu Technicznego w Wiedniu, gdzie podjął pracę jako asystent. Zajmował się zagadnieniami telekomunikacji cyfrowej, modelami cybernetycznymi, teorią informacji, algebrą sieci przełączających — a wszystko to stanowiło wstęp do rozwoju komputerów.

W latach 1954—1959 zebrał grupę studentów, którzy pracowali nad ulepszeniem jednego z pierwszych komputerów w Europie, opartego całkowicie na tranzystorach — Mailüfterl. Przedsięwzięcie to, nie mające oficjalnego poparcia uniwersytetu, było wspomagane głównie przez przemysł i banki; na przykład, wszystkie tranzystory zostały dostarczone przez producenta. Europejskie Biuro Badawcze Armii Stanów Zjednoczonych popierało aplikacyjne prace nad Mailüfterlem, dotyczące budowy kompilatora, przetwarzania danych logicznych i prób w dziedzinie inżynierii oprogramowania. Równolegle prowadzono prace projektowe nad cyfrowym wokoderem i automatyzacją urzędów pocztowych.

Firma IBM zaproponowała utworzenie w Wiedniu laboratorium, mającego kontynuować prace nad Mailüfterlem i wokoderami. Tam też, w 1961 roku, profesor Zemanek przeniósł działalność swojej grupy. Pierwszym produktem opartym na pracy tego laboratorium (rozszerzone i dokończone we Francji, RFN i USA) był system rozpoznawania mowy. Grupa kierowana przez profesora Zemanka rozrosła się do 40 osób. Głównym jej osiągnięciem w latach 1961—1976 było formalne zdefiniowanie języka PL/I. Definicja ta jest zapisana w języku VDL (ang. Vienna definition language), który został rozszerzony do metody VDM (ang. Vienna definition method). W 1964 roku, na Konferencji Roboczej IFIP dotyczącej języków opisu formalnego, zainicjowano szerszą działalność takich metod definiowania.

W 1976 roku profesor Zemanek otrzymał tytuł IBM Fellow. Jego prace dotyczyły teorii projektowania, a szczególnie tzw. architektury abstrakcyjnej, i są stale rozwijane. Profesor Zemanek ma na swoim koncie 360 publikacji (a zapewne do wydrukowania tego artykułu będzie ich już więcej), obejmujących 7 książek, a przy tym niezliczoną liczbę przemówień na konferencjach i seminariach. Otrzymał również wiele nagród. Najwyższym wyróżnieniem jest ustanowienie przez Austriackie Towarzystwo Komputerowe — nagrody im. Heinza Zemanka dla austriackich informatyków.

Wkład profesora Zemanka do prac IFIP, którego był prezydentem w latach 1971—1974, został podsumowany we wrześniowym numerze IFIP Newsletter z 1985 roku. Obecnie profesor Zemanek kończy redagowanie Księgi Jubileuszowej na 25 rocznicę powstania IFIP i przewodniczy komisjom IFIP poświęconym przetwarzaniu informacji i historii tej organizacji.

Ci, którzy mieli przyjemność rozmawiać z profesorem Zemankiem, znają go jako człowieka bardzo wszechstronnego. Wśród wielu swoich zainteresowań wymienia: życie i działalność dziewięciowiecznego perskiego matematyka Al-Chorezmiego, legendę siedmiu śpiących z Efezu i muzykę, głównie komputerową.

(M. K.)

Stała sprzedaż egzemplarzy
MIKROKLANU
prowadzona jest
w Klubie Prasy Technicznej
w Warszawie
ul. Mazowiecka 12, tel. 27-43-65

Terminologia polskiego Logo (2)

Oprócz omówionych w poprzednim numerze nazw operacji arytmetyczno-logicznych i graficznych, w polskim Logo występują też inne grupy nazw. Taką wyraźnie wyodrębnioną grupę stanowią instrukcje operowania słowami i listami.

Już pierwsza nazwa z tej grupy ASCII, oznaczająca operację udostępniania kodu ASCII określonego znaku, nasuwa podejrzenie o brak konsekwencji w dobieraniu polskich odpowiedników nazw angielskich. Jeżeli spolszczono nazwę w pełni symetrycznej do ASCII operacji CHAR, oznaczającej udostępnienie znaku o ustalonej wartości w kodzie ASCII, nazywając ją po polsku ZNAK (ang. character), to tym bardziej należy spolszczyć nazwę ASCII. Skoro uznano za niepotrzebne dodatkowe obciążanie umysłu osób programujących w polskim Logo rozumieniem znaczenia wyrazów angielskich (w tym wypadku — character), to tym bardziej nie należy wprowadzać nazwy ASCII, gdyż wyjaśnienie jej znaczenia wymaga rozszyfrowania skrótu — American Standard Code for Information Interchange — i omówienia kilku jego wyrazów składowych. Przyjmując więc polską nazwę ZNAK na oznaczenie operacji CHAR, należało nadać polską nazwę operacji ASCII, np. KOD. Jeżeli zaś zdecydowano, aby nie zmieniać w polskim Logo nazwy ASCII, to postępując konsekwentnie i logicznie należy przyjąć oryginalną nazwę CHAR na oznaczenie operacji odwrotnej do ASCII.

Co do polskich nazw pozostałych instrukcji dotyczących słów i list nie powinno być wątpliwości. Takie nazwy, jak SŁOWO, LISTA, SŁOWO? i LISTA?, oznaczające instrukcje tworzenia słów i list oraz sprawdzenia, czy dany obiekt jest słowem lub listą, są naturalnymi odpowiednikami nazw angielskich (WORD, LIST, WORDP i LISTP), podobnie jak nazwa ZDANIE, oznaczająca także instrukcję tworzenia listy, SENTENCE.

Ważne instrukcje udostępniania pierwszego i ostatniego elementu listy, FIRST i LAST, skracania listy o pierwszy lub ostatni element, BUTFIRST i BUTLAST, oraz wydłużania listy na początku lub na końcu, FPUT i LPUT, nazwano po polsku, odpowiednio, PIERW, OST, BEZPIERW, BEZOST, NAP i NAK. Sądzę, że tworzenie skrótów w ten właśnie sposób jest pierwszym krokiem do wprowadzenia bełkotu informatycznego. Odpowiednie nazwy polskie powinny mieć pełniejsze brzmienie: PIERWSZY, OSTATNI, BEZPIERW, BEZOST, NAPOCZĄTEK i NAKONIEC.

Pozostałe operacje dotyczące list, COUNT, EMPTYP, ITEM, MEMBERP i NUMBERP, oznaczające odpowiednio badanie długości lub zawartości listy lub słowa, udostępnianie elementu listy, badanie przynależności elementu do listy i badanie, czy obiekt jest liczbą, nazwano po polsku: DŁUGOŚĆ, PUSTE?, ELEMENT, ELEMENT? i LICZBA?. Moje wątpliwości budzi jedynie polska nazwa operacji MEMBERP, która powinna chyba brzmieć NALEŻY?, co bardziej odpowiada składni i semantyce tej operacji.

Dość trudne do zastąpienia spójnym systemem polskich odpowiedników są nazwy operacji dotyczących wprowadzania i wyprowadzania informacji, tj. operacji wejścia-wyjścia: PRINT, READ, SHOW, TYPE i SAVE. Autorzy polskiego Logo nie wybrnęli chyba zbyt szczęśliwie z tej sytuacji, proponując następujące odpowiedniki polskie: PISZ, CZYTAJ, POKAŻ, WPISZ, ZAPISZ. Bez zastrzeżeń można przyjąć jedynie nazwę CZYTAJ, użytą w nazwach instrukcji CZYTAJZNAK i CZYTAJLISTĘ w oryginale — READCHAR i READLIST, oraz — POKAŻ. Pozostałe nazwy polskie są zbyt sobie bliskie fonetycznie i znaczeniowo (jest to praktycznie to samo słowo), aby mogły oznaczać zupełnie różne treści. Na dodatek, w informatyce słowo „zapis" jest zarezerwowane zwyczajowo na oznaczenie angielskiego terminu „write". Wydaje mi się, że jeśli koniecznie trzeba spolszczyć nazwę instrukcji PRINT, to należy ją zastąpić ogólnie przyjętą nazwę DRUKUJ (tak też nazwano w polskim Logo instrukcje PRINTON i PRINTOFF — WŁĄCZDRUK i WYŁĄCZDRUK), a nazwę instrukcji TYPE — nazwą WYŚWIETL. Muszę przy tym zaznaczyć, że osobiście wątpię, aby konieczne było spolszczenie tych właśnie dwóch nazw, używanych na całym świecie w najbardziej rozpowszechnionych językach programowania. Śmiem twierdzić, że każdy uczeń przystępujący do nauki Logo zna już te dwie instrukcje z innych języków. Tak więc, sens spolszczania wszystkiego, jak leci, jest dość wątpliwy.

Angielski wyraz „save" oznacza w informatyce tyle, co „przechowaj" lub „schroń". Ponieważ jednak w Logo nie występuje instrukcja o nazwie WRITE, więc na oznaczenie instrukcji SAVE po polsku można przyjąć proponowaną nazwę ZAPISZ. Jestem jednak przekonany, że można dobrać lepsze odpowiedniki dla nazw instrukcji pochodnych, SAVEALL, SAVED i SAVESCR, na przykład, ZAPISZWSZ, ZAPISZRED i ZAPISZOBR, jeżeli koniecznie muszą to być skróty. Skróty proponowane w polskim Logo różnią się od siebie tylko jedną literą, co może prowadzić do częstych błędów. Ta sama uwaga dotyczy też polskich nazw instrukcji ładowania: LOAD, LOADD i LOADSCR. Uważam, że lepiej byłoby je bardziej zróżnicować, nadając im w dwóch ostatnich wypadkach pełniejsze nazwy, ŁADUJRED, ŁADUJOBR, zamiast, ŁADUJR i ŁADUJO.

Z terminami dotyczącymi drukowania wiąże się jeszcze kilka innych nazw instrukcji, m.in. COPYSCREEN, spolszczona na DRUKOBRAZU, choć lepiej byłoby DRUKUJOBRAZ, i OVER — zastąpiona trafnym odpowiednikiem NADRUK. Operacja o nazwie PO, będącej skrótem określenia „print out" (wydrukuj), służy — mówiąc żargonowo — do tworzenia tzw. listingu procedury, określanego po polsku mianem wydruku. Nie uważam za właściwy zaproponowany w polskim Logo odpowiednik tej nazwy — również PO, jako skrót od POKAŻ — ponieważ wchodzi w kolizję z polskim odpowiednikiem nazwy instrukcji SHOW. Wydaje mi się, że należałoby tu utrzymać nazwę polską zgodną ze znaczeniem wyrazu angielskiego, tzn. WYDRUKUJ. Nazwom instrukcji pochodnych, tzn. PONS, POPS, POTS, POALL, oznaczających wydrukowanie nazw i wartości zmiennych (ang. print out names), treści procedur (ang. print out procedures), nazw procedur (ang. print out titles) oraz nazw procedur z ich treścią i wartościami zmiennych, nadałbym polskie odpowiedniki: WYDRUKNAZWY, WYDRUKPROCEDURY, WYDRUKTYTUŁY, WYDRUKWSZYSTKO (zamiast POWN, POWP, POTP, POW), które oczywiście w praktycznym użyciu należałoby wydatnie skrócić. Związane z wymienionymi operacje usuwania nazw i procedur z pamięci, ERASE, ERN, ERNS, ERPS i ERALL, nazwałbym bardziej rozwlekle niż to uczynili autorzy polskiego Logo, np. USUŃ, USUŃNAZWĘ, USUŃNAZWY, USUŃPROCEDURY, USUŃWSZ (zastępując nic nie mówiące skróty: USN, USWN, USWP, USW).

Dla pozostałych instrukcji służących do wprowadzania i wyprowadzania informacji, .BLOAD, .BSAVE, .DEPOSIT, .EXAMINE, dobrano dość naturalne odpowiedniki polskie: .ŁADUJBIN, .ZAPISZBIN, .UMIEŚĆ i .ZOBACZ, choć w ostatnim wypadku bardziej odpowiednia byłaby nazwa .SPRAWDŹ (instrukcję .CALL, służącą do wywoływania kodu załadowanego instrukcją .BLOAD, nazwano również naturalnie — .WYWOŁAJ). Niestety, dobierając odpowiedniki instrukcji .SERIALOUT i .SERIALIN, służących do przesyłania informacji przez port szeregowy, nie wzięto pod uwagę, że w języku polskim istnieją już przyjęte wyrazy nazywające transmisję w jednym kierunku — nadawaniem, a w kierunku przeciwnym — odbiorem. Dlatego polskie nazwy tych instrukcji powinny brzmieć raczej .NADAJ i .ODBIERZ, zamiast .WYŚLIJ i .PRZYJMIJ. Wydaje mi się też, że lepszą nazwą dla instrukcji ustawiania szybkości transmisji, .SETSERIAL, byłaby nazwa .SZYB-

KOŚĆ, a nie — .TRANSMISJA, jak przyjęto w polskim Logo.

Kończąc omówienie nazw instrukcji związanych z wprowadzaniem i wyprowadzaniem informacji, należy wspomnieć o instrukcji sprawdzania stanu klawiatury, KLAWISZ? (nazwa oryginalna — KEYP), wydawania dźwięków, NUTA (w oryginale — SOUND), i oczekiwania — CZEKAJ (w oryginale — WAIT).

Oprócz wymienionych, typowych grup instrukcji, istnieje szereg instrukcji ogólnych, mających charakter systemowy. Do najczęściej używanych instrukcji tego rodzaju należą TO i END, oznaczające początek i koniec definiowania procedury, dowcipnie nazwane w wersji polskiej wyrazami OTO i JUŻ. Wszelkie poprawki, jak również samo definiowanie procedur, można wykonywać w trybie redagowania, wchodząc do niego za pomocą instrukcji ED, nazwanej po polsku RED, Do redagowania samych nazw służy instrukcja EDNS, nazwana po polsku REDWN, choć mnie osobiście bardziej odpowiadałaby pełniejsza nazwa REDNAZWY.

Instrukcję wyjścia z Logo, BYE, nazwano po polsku — również dowcipnie — DOŚĆ. Natomiast instrukcja TOPLEVEL, oznaczająca powrót do najwyższego poziomu wykonywania poleceń, nie może nazywać się po polsku PRZERWIJ, jak zaproponowano w polskim Logo, ponieważ termin „przerwanie" jest odpowiednikiem angielskiego „interrupt" i oznacza w informatyce ściśle określoną sytuację, z którą wykonanie instrukcji TOPLEVEL nie ma nic wspólnego. Zaproponowałbym tu przyjęcie nazwy WRÓĆ, jako zgodnej z treścią tej instrukcji. Instrukcja powrotu z procedury do procedury bezpośrednio ją wywołującej nazywa się, w wersji angielskiej i polskiej, STOP. Instrukcję OUTPUT, oznaczającą powrót do procedury wywołującej z przekazaniem określonej wartości, nazwano po polsku bardzo trafnie — WYNIK. Mam natomiast szczere wątpliwości, czy instrukcja RUN, będąca poleceniem wykonywania ciągu instrukcji, powinna nazywać się ZRÓB, jak zaproponowano w polskim Logo. Moim zdaniem lepiej byłoby nazwać ją WYKONAJ, co jest zgodne z rzeczywistym sensem tej instrukcji i z rozpowszechnionym w informatyce znaczeniem angielskiego wyrazu „run".

Instrukcje IF i REPEAT mają polskie odpowiedniki JEŻELI i POWTÓRZ, bo trudno wyobrazić sobie lepsze. Nie jestem jednak ostatecznie przekonany, czy nie byłoby lepiej pozostawić nazwy niektórych, bardziej popularnych instrukcji (np. PRINT, TYPE, IF, REPEAT, podobnie jak uczyniono z instrukcją STOP) w oryginalnym brzmieniu.

Nazwy kilku innych instrukcji dotyczących procedur, a mianowicie, TEXT, COPYDEF, DEFINE i DEFINEDP, oznaczających udostępnienie treści procedury, kopiowanie treści, definiowanie procedury i sprawdzanie, czy procedura jest zdefiniowana, dobrano w zasadzie trafnie: TREŚĆ, POWIEL, OKREŚL i OKREŚL?, choć mam poważne wątpliwości, czy angielski wyraz „definition" (i wyrazy pochodne) można zastąpić wyrazem „określenie". Po polsku wyraz „definicja" oznacza dokładne i ścisłe sprecyzowanie znaczenia, natomiast przez „określenie" rozumie się opis właściwości znacznie mniej ścisły i mniej dokładny. Dlatego uważam, że w dwóch ostatnich wypadkach lepszymi nazwami polskimi byłyby: ZDEFINIUJ i ZDEFINIOWANE?

Nazwy zdefiniowane pierwotnie można sprawdzić w Logo instrukcjami PRIMITIVE? i .PRIMITIVES, które nazwano po polsku: PIERWOTNE? i .PIERWOTNE, natomiast pełne informacje o zawartości pamięci można uzyskać za pomocą instrukcji .CONTENTS, którą po polsku nazwano .PAMIĘĆ — moim zdaniem niezbyt trafnie. Lepiej byłoby nadać jej nazwę .ZAWARTOŚĆ, co odpowiada znaczeniu angielskiego wyrazu „contents" i faktycznej treści odpowiedniego pojęcia. Instrukcje dotyczące gospodarki pamięcią, .RESERVE, .RESRVED, NODES i RECYCLE, nazwano w polskim Logo następująco: .ZAJMIJ, .ZAJĘTE, WOLNE i ODŚMIEĆ. Sądzę, że nie ma konieczności, aby wprowadzać do języka tę ostatnią nazwę. Angielski wyraz „recycle" znaczy w tym przypadku „przywróć", lecz z uwagi na symetrię względem pozostałych trzech instrukcji tej grupy instrukcję RECYCLE nazwałbym po polsku ZWOLNIJ, gdyż jej działanie polega właśnie na zwolnieniu pewnego obszaru pamięci.

Instrukcje nadawania wartości zmiennym i sprawdzania tych wartości, MAKE i THING, nazwano po polsku bardzo trafnie, PRZYPISZ i WARTOŚĆ. Sądzę natomiast, że instrukcja NAMEP, służąca do sprawdzenia, czy określona zmienna (obiekt) ma wartość, nazwana po polsku JEST?, powinna nazywać się zgodnie z treścią odpowiedniej operacji, np. MAWARTOŚĆ?

Pełne zestawienie proponowanych odpowiedników nazw instrukcji Logo przedstawimy w następnym numerze.

JANUSZ ZALEWSKI

WARUNKI PRENUMERATY W 1987 ROKU

Prenumeratorzy zbiorowi — jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę, dokonując wpłaty wyłącznie na blankiecie „polecenie przelewu" rozszerzonym dla potrzeb Wydawnictwa o część dotyczącą zamówienia. Blankiety te będą dostarczane przez Zakład Kolportażu.

Prenumeratorzy indywidualni — osoby fizyczne zamawiają prenumeratę, dokonując wpłaty w UPT lub NBP na blankiecie Wydawnictwa lub blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamówionych egzemplarzy oraz wartość wpłaty.
Wpłacać należy na konto: NBP III Oddział Warszawa 1036-7490--139-11.

Prenumerata ulgowa — przysługuje wyłącznie osobom fizycznym — członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczątką Koła SNT, wyższej uczelni lub szkoły.
Sposób zamawiania prenumeraty ulgowej jest taki sam jak dla prenumeraty indywidualnej.

Prenumerata ze zleceniem wysyłki za granicę — zamawia się tak jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy.
Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Wpłaty na prenumeratę przyjmowane są w terminach:
— do 10 listopada na I kwartał, I półrocze i cały rok następny,
— do 28 lutego na II, III i IV kwartał i II półrocze,
— do 31 maja na III, IV kwartał i II półrocze,
— do 31 sierpnia na IV kwartał.

UWAGA!

Wpłaty na dwumiesięczniki przyjmowane są na okresy półroczne i roczne.

Informacji o prenumeracie udziela — Zakład Kolportażu Wydawnictwa NOT SIGMA, ul. Bartycka 20, 00-716 Warszawa lub skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 wew. 249, 293, 297, 299 oraz 40-30-86 i 40-35-89.

Egzemplarze archiwalne czasopism — można nabywać za gotówkę w Klubie Prasy Technicznej w Warszawie, ul. Mazowiecka 12, tel. 27-43-65 lub zamówić w Dziale Handlowym Wydawnictwa NOT SIGMA skr. poczt. 1004, 00-950 Warszawa, tel. 40-37-31, na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Cena prenumeraty wg cennika					
kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
450	150	900	300	1800	600

■ International Business Machines Corp — oferuje swym pracownikom dobrowolne przejście na wcześniejszą emeryturę. Nowe przepisy dodają pięć lat do posiadanego stażu pracy. Przewiduje się, że w 1987 roku około czterech tysięcy pracowników IBM skorzysta z tej propozycji, a łącznie z wciąż ograniczonym naborem spowoduje to redukcję personelu o 8 tysięcy.

Na koniec 1985 roku firma IBM miała w Stanach Zjednoczonych 242 241 pracowników, a na 1986 roku przewidziano zmniejszenie tej liczy o cztery tysiące.

Zachęta do wcześniejszej emerytury wynikła z potrzeby zmniejszenia kosztów utrzymania przedsiębiorstwa. Pomoże to także utrzymać tradycję pełnego zatrudnienia, gdyż firma nie zwalniała pracowników w trudnych dla siebie okresach. Ocenia się, że nadmiar pracowników spowodowany był przesadną prognozą produkcyjną. Przy ogólnych trudnościach przemysłu komputerowego, IBM odczuwa zwiększoną konkurencję w zakresie komputerów osobistych i średniej wielkości. Zyski tej firmy w II kwartale 1986 roku spadły o 8%, a rezultaty za cały rok już po raz drugi pogorszyły się. Akcje IBM kosztowały w pierwszej połowie września 1986 roku 139 dolarów w porównaniu ze 162 dolarami na początku roku. (IR).

★

■ Zmiany w amerykańskim systemie podatkowym pociągają za sobą konieczność zmiany oprogramowania opracowanego dotychczas do tego celu. Zajmuje się tym firma Accountants Microsystems Inc., której przedstawiciel stwierdził, że będą one przeprowadzone we właściwym czasie i bez dodatkowych kosztów. Dodał także, że dzięki systemom elektronicznego druku wprowadzonym w końcu 1984 roku, koszt publikacji jednej strony zmniejszył się z 10—15 dolarów do 1—3 dolarów, a sam proces jest trzykrotnie krótszy. Systemy te łączą przetwarzanie tekstów z możliwościami graficznymi, co pozwala łączyć rysunki, tabele, nagłówki i tekst. Wszystkie kroki mogą być wykonywane z użyciem monitorów ekranowych. Wynik, niezależnie, czy dotyczy to strony czy całej książki, może być elektronicznie przesłany do urządzenia składającego lub drukarki. Tylko złożone wydruki kolorowe i oprawa są wykonywane poza systemem, natomiast wszystkie dokumenty wewnętrzne mogą być wytwarzane w ramach systemu wydawniczego.

Obecnie takie systemy bardzo się rozwijają. W 1985 roku sprzedano je za przeszło pół miliarda dolarów, natomiast dwa lata wcześniej prawie ich nie było. W systemy elektronicznego druku inwestuje się znaczne sumy. Zwiększa się też wartość oferowanych wyrobów. Przykładowo, firma Interleaf Inc., opracowująca oprogramowanie do systemów elektronicznego druku, ogłosiła w czerwcu 1986 roku ofertę za 24,6 mln dolarów, a Xyvision Inc. zwiększyła wartość oferowanych wyrobów o 18,9 mln. W lipcu firma IBM ogłosiła utworzenie działu opracowującego sprzęt i oprogramowanie do wspomaganych komputerowo wydawnictw (CAP, ang. computer aided publishing). We wrześniu ubiegłego roku ponad 150 firm opracowywało oprogramowanie do tych celów. Niektóre analizy rynku przewidują w związku z tym bankructwo słabszych firm, ale D. Goodstein z Interconsultant Inc. uważa, że jest to najszybciej rozwijająca się dziedzina przemysłu komputerowego, zwłaszcza jeśli chodzi o systemy oparte na komputerach osobistych w cenie 10—40 tys. dolarów. Obecnie istnieje 40—50 tys. takich systemów i ich liczba powiększa się o tysiąc tygodniowo, w ciągu lat 1987—1988 będzie wzrastała o 2,5—3 tys. tygodniowo.

Duże zyski osiąga firma Apple Computer Inc, której komputer Macintosh i drukarka laserowa Laserwriter wyjątkowo pasuje do tego zastosowania. Programy na te urządzenia opracowały firmy Manhattan Graphics i Boston Software Publishers.

Zakupy systemów elektronicznego druku przez handlowców osiągnęły wartość prawie 900 mln dolarów w 1986 roku i ponad 1,3 mld dolarów w 1987 roku, a przez użytkowników — odpowiednio prawie 400 mln i ponad 700 mln dolarów. (IR).

★

■ Firma Apple Computer Inc. podała 15 września 1986 roku parametry nowej wersji komputera Apple II o nazwie Apple II-GS oraz obniżyła cenę komputera Macintosh o 400 dolarów, tj. do 2199 dolarów. Komputer Apple II-GS jest szybszą i bardziej elastyczną wersją komputera tej rodziny. Łączy on możliwości rozszerzania konfiguracji jaką ma komputer Apple II, z dobrą rozdzielczością grafiki Macintosha. Koszt głównej części komputera bez monitora i stacji dysków wynosi 999 dolarów, a całego systemu 1800 dolarów.

Specjaliści oceniają, że nowy komputer umocni pozycję firmy Apple na rynku edukacyjnym, gdyż ok. 60% komputerów używanych w szkołach pochodzi z tej firmy. Ponadto niektóre jego parametry mogą być interesujące dla posiadaczy komputerów domowych.

Przedstawiciel firmy poinformował, że przy opracowywaniu komputera II-GS dążono do poprawienia grafiki i dźwięku, szybszego przetwarzania i powiększenia pojemności pamięci, a jednocześnie zachowania kompatybilności z dotychczasowym oprogramowaniem i urządzeniami peryferyjnymi.

Obniżono też cenę drukarki laserowej o 1000 dolarów do wartości 4999 dolarów. (IR).

★

■ Mnożą się spekulacje, nie potwierdzone jeszcze przez przedsiębiorstwo Honeywell, mające swą centralę w Minneapolis, że zamierza ona sprzedać swój dział komputerowy japońskiej firmie NEC i francuskiej Bull, a dział obronny — firmie Boeing Corp. Ogłoszono jedynie, że podpisano umowę na współpracę z firmą NEC Corp. Dotyczy ona sprzedaży superkomputerów SX-1 i SX-2 w USA i Kanadzie. Firma japońska NEC jest największym producentem układów półprzewodnikowych w swoim kraju i dostarcza procesory do urządzeń Honeywella, a także sprzedaje komputery tej firmy w Japonii. Firma Bull, której 9% akcji należy do Honeywella, sprzedaje wyroby obu firm w 75 krajach.

Ostatnio pogorszyła się również sytuacja finansowa firmy Honeywell. Zysk spadł w pierwszej połowie 1986 roku o 37%, tj. do 63,2 mln dolarów, przy sprzedaży o wartości globalnej 3,2 mld dolarów. W 1985 roku komputery stanowiły 35% obrotów firmy, wynoszących 6,62 mld dolarów. Pozostała część, to elektronika wojskowa oraz systemy sterowania zakładami zbrojeniowymi, budowlanymi i urządzeniami zabezpieczającymi. W ramach zwiększenia wydajności przedsiębiorstwo przewidywało do końca 1986 roku redukcję o 4000 osób z personelu liczącego 75 tys. (IR).

★

■ Czasopismo Datamation z sierpnia 1986 roku podaje listę przedsiębiorstw europejskich, o największych dochodach w dziedzinie przetwarzania danych. Dochody te są podane w milionach dolarów.

1.	Siemens AG	RFN	3265
2.	C. Olivetti	Włochy	2638
3.	Bull	Francja	1795
4.	N. V. Philips	Holandia	1366
5.	Nixdorf Computer AG	RFN	1340
6.	STC	W. Brytania	1331
7.	L. M. Ericsson	Szwecja	1233
8.	Companie Generale d'Electricite	Francją	479
9.	British Telecom	W. Brytania	455
10.	Volkswagen AG	RFN	453

(IR)

★

■ Minister przemysłu elektrotechnicznego i elektronicznego NRD Felix Meier ujawnił ostatnio, że plany kombinatu Robotron przewidują wyprodukowanie w bieżącym pięcioleciu co najmniej 160 000 komputerów osobistych i biurowych, 1950 minikomputerów oraz 670 dużych komputerów, a oprócz tego znacznie rozszerzonego asortymentu urządzeń peryferyjnych. W dziedzinie zastosowań punktem ciężkości będą systemy CAD/CAM. W tym celu do 1990 r. planuje się wdrożenie w przedsiębiorstwach 5800 tego typu systemów. Opierać się one będą na powszechnie dostępnych uogólnionych i zapewniających łatwą adaptację rozwiązaniach programowych. Zastosowania CAD/CAM powinny zapewnić nie tylko znaczny wzrost wydajności pracy konstruktorów, projektantów i twórców form przemysłowych, ale również doprowadzić do zmniejszenia materiało- i energochłonności wyrobów oraz zwiększenia ich walorów użytkowych. Szacuje się, że każde z tych zastosowań powinno umożliwić przesunięcie do innych zadań od 3 do 20 pracowników, co w sytuacji ostrego deficytu kadr w NRD jest szczególnie istotnym zadaniem polityki gospodarczej. (WK)

Madey J.: Problematyka systemów operacyjnych na przykładzie systemu Unix (1)

INFORMATYKA 1987, nr 6, s. 1

Charakterystyka ewolucji podstawowych cech i funkcji współczesnych systemów operacyjnych na przykładzie rozwiązań systemu Unix.

Madey J.: Unix as an example of operating system (1)

INFORMATYKA 1987, No. 6, p. 1

Characteristics of evolution of contemporary operating systems features and functions with example of Unix system solutions.

Madey J.: Betriebssystemenproblematik an Beispiel des Unix-Systems (1)

INFORMATYKA 1987, Nr. 6, S. 1

Eine Charakteristik der Evolution von Merkmalen und Funktionen der heutigen Betriebssysteme an Beispiel von Lösungen des Unix-Systems.

Kowalski R.: Logika w systemach ekspertowych (2)

INFORMATYKA 1987, nr 6, s. 5

Druga część artykułu zawierająca charakterystykę podejścia do rozwiązania problemów w systemie ekspertowym na przykładzie rozwiązywania równań matematycznych.

Kowalski R.: Logic in expert systems (2)

INFORMATYKA 1987, No. 6, p. 5

Second part of the paper, which includes discussion of problem solving in expert systems on example of mathematical equations solving.

Kowalski R.: Logik in Expertensystemen (2)

INFORMATKA 1987, Nr. 6, S. 5

Zweiter Teil des Artikels, der eine Charakteristik der Einstellung zur Problemlösung in Expertensystem an Beispiel von Lösung der mathematischen Gleichungen umfasst.

Szkaradnik Z., Petrykowski W.: Grafika żółwia w języku Forth (2)

INFORMATYKA 1987, nr 6, s. 8

Druga część artykułu zawierająca przykłady możliwości praktycznego wykorzystania oraz rozbudowy procedur graficznych w języku Forth.

Szkaradnik Z., Petrykowski W.: Turtle graphics in Forth language (2)

INFORMATYKA 1987, No. 6, p. 8

Second part of the paper with examples for practical using and exensions of graphic procedures in Forth language.

Szkaradnik Z., Petrykowski W.: Schildkrötegrafik in Forth-Sprache (2)

INFORMATYKA 1987, Nr. 6, S. 8

Zweiter Teil des Artikels mit Beispielen der praktischen Ausnutzungs- und Erweiterungsmöglichkeiten der grafischen Prozeduren in Forth-Sprache.

Morzy T.: Synchronizacja transakcji w systemach rozproszonych baz danych (2)

INFORMATYKA 1987, nr 6, s. 11

Druga część artykułu, zawierająca charakterystykę trzech podstawowych metod synchronizacji transakcji w systemach rozproszonych baz danych oraz omówienie aktualnych tendencji w zakresie tworzenia algorytmów synchronizacji.

Morzy T.: Transaction synchronization in distributed data base systems (2)

INFORMATYKA 1987, No. 6, p. 11

Second part of the paper, which includes characteristics of three basic methods for transactions synchronization in distributed data base systems as well as discussion of current trends in synchronization algorithms.

Morzy T.: Transaktionssynchronisation in verteilten Datenbanksystemen (2)

INFORMATYKA 1987, Nr. 6, S. 11

Zweiter Teil des Artikels, der eine Charakteristik von drei Grundmethoden der Transaktionssynchronisation in verteilten Datenbanksystemen, sowie eine Besprechung von heutigen Trends im Bereich der Synchronisationsalgorithmengestaltung, umfasst.

Ratka C.: Adaptacja drukarki wierszowej DW 204-2 do pracy z minikomputerem SM-4

INFORMATYKA 1987, nr 6, s. 14

Charakterystyka sprzętowej adaptacji drukarki wierszowej DW-204-2 do współpracy z systemem minikomputerowym SM-4, jako przykład możliwości wzajemnego przystosowania urządzeń zewnętrznych o różnych standardach.

Ratka C.: DW 204-2 line printer adaptation for SM-4 minicomputer

INFORMATYKA 1987, No. 6, p. 14

Characteristics of DW 204-2 line printer hardware adaptation for cooperation with SM-4 minicomputer system as an example for mutual adaptation of peripheral devices of different standards.

Ratka C.: Anpassung des DW 204-2-Paralleldruckers zur Mitwirkung mit SM-4-Kleincomputer

INFORMATYKA 1987, Nr. 6, S. 14

Eine Charakteristik der Hardwareanpassung von DW 204-2-Paralleldrucker zur Mitwirkung mit SM-4-Kleincomputer als Beispiel von Anpassungsmöglichkeit der Peripherieanlagen mit verschiedenen Standarden.

Zalewski J.: Concurrent DOS (2)

INFORMATYKA 1987, nr 6, s. 17

Druga część charakterystyki systemu operacyjnego Concurrent DOS.

Zalewski J.: Concurrent DOS (2)

INFORMATYKA 1987, No. 6, p. 17

Second part of characteristics of the Concurrent DOS operating system.

Zalewski J.: Concurrent DOS (2)

INFORMATYKA 1987, Nr. 6, S. 17

Zweiter Teil einer Charakteristik von Concurrent DOS-Betriebssystem.

Zjazd użytkowników komputerów SHARP

Klub Użytkowników Komputerów Osobistych **MIKROKOMPUTER** w Łodzi, działający w ramach Ogólnopolskiej Federacji Klubów Komputerowych przy Radzie Naczelnej Zrzeszenia Studentów Polskich informuje, że w dniach 17—18 października 1987 roku w Łodzi, odbędzie się I Zjazd Użytkowników Komputerów SHARP. Celem Zjazdu jest konsolidacja środowiska, wymiana informacji i doświadczeń, przedstawienie własnych osiągnięć członków sekcji SHARP, popularyzacja nowych rozwiązań sprzętowych i programowych. W programie — wykłady i referaty dotyczące użytkowania komputera SHARP MZ.

Do wzięcia udziału w Zjeździe zaprasza się również użytkowników nietypowych komputerów z mikroprocesorem Z-80, gdyż ich też powinny zainteresować poruszone zagadnienia. Istniejąca od 1985 roku, jedyna w kraju sekcja użytkowników SHARPA oraz również jedyna w kraju sekcja użytkowników komputerów nietypowych z procesorem Z-80, działająca w ramach Klubu, zrzeszają w swych szeregach posiadaczy takich komputerów z terenu całej Polski.

Informacje na temat Zjazdu można uzyskać w Biurze Organizatora:

DDK Łódź-Polesie, ul. 1 Maja 87, tel. 33-08-02
lub bezpośrednio u osób koordynujących przygotowania do Zjazdu:
wiceprezes Klubu, szef sekcji SHARP — JERZY GARLICKI, ul. A. Struga 4, tel. 32-57-83
wiceprezes Klubu, szef sekcji NIETYPOWYCH — **ROMUALD ŻYŁŁA, ul. Zachodnia 12, tel. 57-75-06.**

Przedsiębiorstwo Projektowania

PROMEL

oferuje do sprzedaży:

- **1 dalekopis typu TELETYPE** przystosowany do współpracy z minikomputerem WANG
  w cenie ok. 200 tys. zł

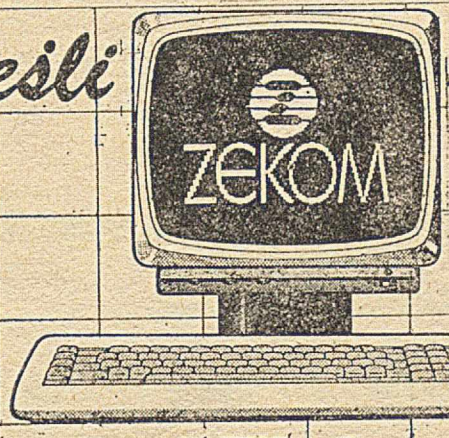
- **10 dysków wymiennych twardych** o pojemności 5 Mb
  do minikomputera WANG

Szczegółowych informacji udziela:

PROMEL

ul. Kościuszki 1
44-100 Gliwice
tel. 32-15-25 wew. 117

Jeśli interesuje Państwa



* Profesjonalny sprzęt o wysokiej jakości, niezawodny w eksploatacji
* sprawny serwis
* krótkie terminy dostaw, lub dostawy natychmiastowe

TO WYROBY ZEKOMU SĄ DO PAŃSTWA DYSPOZYCJI

Zakład Elektroniki Komputerowej

oferuje terminale ekranowe:

MV 2580 Standard VT 52 firmy DEC / odpowiednik MERA 7953. Przeznaczony do pracy w systemach komputerowych wyposażonych w kanał transmisji V 24 lub pętli prądowej 20/60 mA – jako końcówka zdalnego dostępu.

MV 2581 Odpowiednik MERA 7911 N.
Przeznaczony do pracy w systemach komputerowych ODRA 1300 wyposażonych w jednostkę sterującą MERA 7802.

MV 2582E Odpowiednik terminala typu 7181/2 firmy ICL.
Przeznaczony do pracy w systemach komputerowych ODRA 1300, ICL 1900, ICL 2900, ICL system 4.

MR 1240 Odpowiednik MERA 7951.
Przeznaczony do wprowadzania danych do systemu MERA 9150 lub systemu REDIFON.

ZAKŁAD ELEKTRONIKI KOMPUTEROWEJ ul. Makowa 8, 91-480 Łódź tel. 34 30 49

EO/307/87

KONIEC TWOICH PROBLEMÓW

Nareszcie wszystkie potrzebne Ci dane dotrą na czas
w przejrzystej formie tabel i wykresów.

Pakiet oprogramowania na mikrokomputery 16-bitowe

MEGA — BANK

to nowe MEGA możliwości jakie otwierają się przed
Twoim przedsiębiorstwem.

Wyobraź sobie

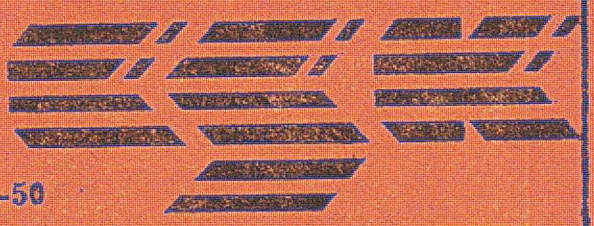
MILIARD

rekordów, które możesz
zapełnić według własnych potrzeb i uznania.

Miliard kooperantów, miliard pracowników, miliard produktów
— z tym wszystkim nasz MEGA-BANK poradzi sobie bez trudu.

System jest łatwy w obsłudze i opracowany w języku polskim.

Gwarantujemy satysfakcję!



COMPUTER STUDIO KAJKOWSCY
PROFESJONALNE OPROGRAMOWANIE MIKROKOMPUTERÓW
ul. Balladyny 3B, 81-524 Gdynia, tel.: 29-00-18, 24-01-50

EO/81/K/43